

Querying languages over sliding windows

Moses Ganardi, Danny Hucce and Markus Lohrey

Universität Siegen, Germany

February 16, 2017

Abstract

We study the space complexity of querying languages over data streams in the sliding window model. The algorithm has to answer at any point of time whether the content of the sliding window belongs to a fixed regular language. For regular languages, a trichotomy is shown: For every regular language the optimal space requirement is asymptotically either constant, logarithmic, or linear in the size of the sliding window. Moreover, given a DFA for the language, one can check in nondeterministic logspace (and hence polynomial time), which of the three cases holds. For context-free languages the trichotomy no longer holds: For every $c \geq 1$ there exists a context-free language L_c for which the optimal space required to query L_c in the sliding window model is asymptotically $n^{1/c}$. Finally, it is shown that all results also hold for randomized (Monte-Carlo) streaming algorithms.

This paper is an extended version of the conference version [17].

1 Introduction

Streaming algorithms process an input sequence $a_1a_2 \cdots a_m$ from left to right and have at time t only direct access to the current symbol a_t . Such algorithms received a lot of attention in recent years, see [1] for a general reference. Two variants of streaming algorithms can be found in the literature:

- In the *standard model* the algorithm computes at each time instant t a value $f(a_1 \cdots a_t)$ that depends on the whole history.
- In the *sliding window model* the algorithm computes at each time instant t a value $f(a_{t-n+1} \cdots a_t)$ that depends on the n last symbols (we should assume $t \geq n$ here). The value n is also called the *window size*.

For many applications, the sliding window model is more appropriate. Quite often data items in a stream are outdated after a certain time, and the sliding window model is a simple way to model this. The typical application is the

analysis of a time series as it may arise in medical monitoring, web tracking, or financial monitoring. In all these applications, the most recent data items are more important than older ones.

A general goal in the area of sliding window algorithms is to avoid the explicit storage of the whole window, and, instead, to work in considerably smaller space, e.g. polylogarithmic space. In the seminal paper of Datar et al. [11], where the sliding window model was introduced, the authors prove that the number of 1's in a 0/1-sliding window of size n can be maintained in space $\frac{1}{\varepsilon} \cdot \log^2 n$ if one allows a multiplicative error of $1 \pm \varepsilon$. A matching lower bound is provided as well in [11]. Other algorithmic problems that were addressed in the extensive literature on sliding window streams include the computation of statistical data (e.g. computation of the variance and k -median [4], and quantiles [3]), optimal sampling from sliding windows [9], database querying (e.g. processing of join queries over sliding windows [19]) and graph problems (e.g. checking for connectivity and computation of matchings, spanners, and minimum spanning trees [10]). The reader can find further references in the surveys [1, Chapter 8] and [8]. Another natural problem, whose investigation has so far been surprisingly neglected for the sliding window model, is the membership problem for a language or, equivalently, the computation of Boolean queries on the sliding window. In its general form, one fixes a language L over the alphabet of the data stream, and asks for an algorithm that can check at any time whether the content of the sliding window belongs to L . In this paper, we are mainly interested in the case, where L is a regular language.

Note that in the standard streaming model, it is trivial to solve the membership problem for a regular language L in constant space. For an input stream $a_1 a_2 \dots a_m$ the algorithm simply runs a deterministic finite automaton for L and only stores the current state (which needs constant space since we assume the automaton to be fixed and not part of the input). This obvious fact might explain why the membership problem for regular languages in the streaming model has not received any attention so far. In contrast, there exist papers that deal with membership problems for (restricted classes of) context-free languages in the standard streaming model, see the paragraph on related work below.

Main results of the paper. Note that in the sliding window model the above algorithm (simulation of a DFA on the data stream) does not work. The problem is the removal of the left-most symbol from the sliding window in each step. A naïve approach is to store the whole window in $O(n)$ bits and simulate the DFA on this word. In fact, there exist very simple languages L for which this is the best possible solution in order to be able to decide at any point of time whether the current content of the sliding window belongs to L . An example is the language $a\{a, b\}^*$ of all words that start with a .

The point is that by repeated checking whether the sliding window content belongs to $a\{a, b\}^*$, one can recover the exact content of the sliding window, which implies that every sliding window algorithm for querying $a\{a, b\}^*$ has to use n bits of storage (where n is the window size). The main result of this paper is a trichotomy: The optimal space needed for querying a regular language L in the sliding window model falls into three classes with respect to its growth rate: constant space, $O(\log n) \setminus o(\log n)$, and $O(n) \setminus o(n)$, where n is the window size.¹ We characterize the regular languages by its optimal growth rate algebraically in terms of the syntactic homomorphism and the left Cayley graph of the syntactic monoid of a regular language. The precise characterizations are a bit technical and will be presented in Section 5.

The sliding window model we have talked about so far is also known as the *fixed-size model*, since the sliding window has a fixed size n . In the literature there exists a second model as well which is known as the *variable-size model*, see e.g. [3]. In this model, the arrival of new data items and the expiration of old items can happen independently, which means that the sliding window can grow and shrink. We also determine the space complexity of querying a regular language for the variable-size model. Again, we prove the same trichotomy as above (constant space, logarithmic space, and linear space). In contrast to the fixed-size model, only the trivial languages \emptyset and Σ^* are streamable in constant space in the variable-size model. On the other hand, it turns out that the regular languages streamable in logarithmic space coincide for the fixed-size and the variable-size model. We also show that these languages are exactly those languages that are reducible with a Mealy machine (working from right to left) to a regular language of polynomial growth. The regular languages of polynomial growth are exactly the bounded regular languages [28]. A language L is *bounded* if there are words w_1, w_2, \dots, w_n such that $L \subseteq w_1^* w_2^* \dots w_n^*$.

In Section 6 we extend the lower bounds from Section 5 to randomized streaming algorithms, i.e., streaming algorithms that have access to random numbers. We consider Monte-Carlo streaming algorithms for the above described fixed-size sliding window model. A Monte-Carlo algorithm is only required to work correctly with large probability (we set this probability to $2/3$) for every input. “Working correctly” here means that while reading the input from left to right, the algorithm gives at each time instant a correct answer. This definition of correctness is used in most of the work on randomized sliding window algorithms, see e.g. [7, 11]. A Monte-Carlo streaming algorithm works in space $s(n)$ (where n is the window size) if for every input, the average number of bits stored by the algorithm while reading the input is bounded by $s(n)$, where the average is computed with respect to the random choices of the algorithm. We show that for every

¹Note that $g \in O(f) \setminus o(f)$ means that there are constants c_1 and c_2 such that (i) $g(n) \leq c_2 \cdot f(n)$ for all large enough n and (ii) $g(n) \geq c_1 \cdot f(n)$ for infinitely many n .

language L , the optimal space needed to query L in the fixed-size streaming model only decreases by a constant factor when going from the deterministic to the Monte-Carlo setting.

Section 7 addresses the problem of checking whether for a given deterministic finite automaton \mathcal{A} the optimal space for querying $L(\mathcal{A})$ in the sliding window model is constant, logarithmic or linear. We prove that this problem can be solved in nondeterministic logarithmic space (NL) and hence in deterministic polynomial time, by reducing it to the emptiness problem for nondeterministic one-counter automata, which is NL-complete [24].

A very natural question is, whether our space trichotomy (constant, logarithmic or linear space) also holds for natural generalizations of the regular languages. An obvious candidate is of course the class of context-free languages. In Section 8 we prove that for every natural number $c \geq 1$ there exists a context-free language L_c , where the optimal space required to query L_c in the fixed-size sliding window model is $O(n^{1/c}) \setminus o(n^{1/c})$.

The above mentioned variable-size sliding window model can be seen as a dynamic string data structure problem: A string is dynamically modified using the following operation: Pop a letter from the left end of the structure (**leftpop**), and push a given letter a to the right end of the string (**rightpush**(a)). One is looking for a succinct representation of the dynamic string arising from a sequence of these operation, that allows to answer membership queries with respect to the language L . Natural variations of this problem are obtained by considering also the operations of popping a letter at the right end of the string (**rightpop**) and pushing the letter a at the left end of the string (**leftpush**(a)). For instance, a stack is obtained by allowing **rightpop** and **rightpush**(a) for all symbols a . In Section 9 we consider all reasonable combinations \mathcal{M} of **leftpop**, **rightpop**, **leftpush**(a) and **rightpush**(a). Reasonable means for instance that all **leftpush**(a)-operations or all **rightpush**(a)-operations are allowed, so that arbitrary strings can be built up. For every regular language L we determine the optimal space needed to answer membership queries for L and a dynamic string that is modified by an adversary using the operations from \mathcal{M} . It turns out that in all cases, the optimal space is asymptotically either constant, logarithmic, or linear with respect to the current length of the dynamic string.

Related work. In [5] the authors consider the problem of membership checking for various subclasses of context-free languages in the standard streaming model (where the whole history is checked for membership). For deterministic linear languages, a randomized streaming algorithm is presented which works in space $O(\log n)$ and has an inverse polynomial one-sided error. On the other hand, a visibly pushdown language L exists, for which every randomized streaming algorithm with an error probability $< 1/2$ must use space $\Omega(n)$ [5]. In this context one should also mention the

work on XML stream validation [25]. There, the goal is to check by a finite automaton, whether an XML tree, represented by a string of opening and closing tags, satisfies a given DTD (document type definition).

A variant of the standard streaming model for language recognition was recently proposed by Fijalkow in [14]. In [15], the authors present a streaming ε -property tester for visibly pushdown languages working in space $(\log n)/\varepsilon)^{O(1)}$. A streaming ε -property tester for a language L is a streaming algorithm recognizing a language under the property testing approximation: it must distinguish inputs of the language from those that are ε -far from it.

Streaming a language L in the standard model is also related to the concept of *automaticity* [26]. For a language $L \subseteq \Sigma^*$, the automaticity A_L of L is function $n \mapsto A_L(n)$, where $A_L(n)$ is the minimal number of states of a DFA \mathcal{A} such that for all words w of length at most n : $w \in L$ if and only if $w \in L(\mathcal{A})$. In other words: $L(\mathcal{A})$ and L coincide on all words of length at most n . Clearly, every regular language L has constant automaticity. Karp [22] proved that for every non-regular language L , $A_L(n) \geq (n+3)/2$ for infinitely many n . This easily implies that for every non-regular language L , membership checking in the standard streaming model is not possible in space $o(\log n)$.

As mentioned above, one may consider our streaming algorithms as algorithms for testing membership of a dynamic word in a language L , where the update operations are restricted. In the variable-size model, these updates are popping the left most symbol and pushing an a at the right end. Membership testing algorithms for regular languages that allow the replacement of the symbol at a specified position were studied in [16]. The focus of [16] is on the cell probe complexity of updates and membership queries.

2 Preliminaries

2.1 General notations

For a function $f : \mathbb{N} \rightarrow \mathbb{N}$, we use the standard Landau notations $O(f)$, $\Omega(f)$, $o(f)$ and $\Theta(f)$.

For an equivalence relation \equiv on a set A and $a \in A$ we denote with $[a]_{\equiv} = \{b \in A : b \equiv a\}$ the equivalence class containing a . The set of all equivalence classes is $A/\equiv = \{[a]_{\equiv} : a \in A\}$.

2.2 Words and languages

We assume that the reader is familiar with the basic notions of formal languages, in particular regular languages, see e.g. [21] for more details. Let Σ be an alphabet of symbols, which in most cases will be finite. With ε we denote the empty word. For a word $w = a_1 \cdots a_m \in \Sigma^*$ of length $|w| = m$ we define $w[i] = a_i$ and $w[i : j] = a_i \cdots a_j$ if $i \leq j$ and $w[i : j] = \varepsilon$ if $i > j$.

We define $w[i:] = w[i:m]$ and $w[:j] = [1:j]$. Let $\Sigma^n = \{w \in \Sigma^* : |w| = n\}$, $\Sigma^{\leq n} = \{w \in \Sigma^* : |w| \leq n\}$, and $\Sigma^{\geq n} = \{w \in \Sigma^* : |w| \geq n\}$. A word $v \in \Sigma^*$ is a *prefix* (resp., *suffix*) of the word w if there exists a word $u \in \Sigma^*$ such that $w = vu$ (resp., $w = uv$). With $\text{prefix}(w)$ we denote the set of all prefixes of w . For a word $w = a_1 a_2 \cdots a_m$ let $\text{rev}(w) = a_m \cdots a_2 a_1$ denotes the word w read from right to left.

For a language $L \subseteq \Sigma^*$ over a finite alphabet Σ one defines the *Myhill-Nerode equivalence* \sim_L as follows, where $u, v \in \Sigma^*$: $u \sim_L v$ if and only if for all $x \in \Sigma^*$: $ux \in L$ if and only if $vx \in L$. It is easy to see that \sim_L is an equivalence relation and a right congruence, i.e., $u \sim_L v$ implies $ux \sim_L vx$ for all $x \in \Sigma^*$.

2.3 Finite automata

A *deterministic automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a set of states, Σ is the finite input alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states. We inductively extend δ to $\delta : Q \times \Sigma^* \rightarrow Q$ by $\delta(q, \varepsilon) = q$ and $\delta(q, wa) = \delta(\delta(q, w), a)$ for all $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$. Instead of $\delta(q_0, w)$, we also write $\mathcal{A}(w)$. The language accepted by \mathcal{A} is $L(\mathcal{A}) = \{w \in \Sigma^* : \mathcal{A}(w) \in F\}$. We can always assume that all states $q \in Q$ are reachable from q_0 .

If Q is finite, then \mathcal{A} is called *deterministic finite automaton*, or short DFA. A language $L \subseteq \Sigma^*$ is called *regular* if it is accepted by a DFA. The famous Myhill-Nerode theorem states that $L \subseteq \Sigma^*$ is regular if and only if Σ^*/\sim_L is finite. In that case $|\Sigma^*/\sim_L|$ is the number of states of the smallest DFA that accepts L (which is unique up to renaming of states). This smallest DFA can be constructed from any DFA for L by Hopcroft's algorithm, which merges equivalent states. Two states $p, q \in Q$ of the DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ are equivalent if for all $w \in \Sigma^*$ we have $\delta(p, w) \in F$ if and only if $\delta(q, w) \in F$. In a minimal DFA, two distinct states p and q are not equivalent, i.e., there exists a word $w \in \Sigma^*$ such that $\delta(p, w) \in F$ if and only if $\delta(q, w) \notin F$.

2.4 Monoids

Our query algorithms for regular languages make use of the description of regular languages by finite monoids; see e.g. the textbook [27] for more details. A *monoid* is a set M together with an associative operation $\cdot : M \times M \rightarrow M$ and an element $1 \in M$ satisfying $1 \cdot x = x \cdot 1 = x$ for all $x \in M$. A function $h : M \rightarrow N$ between two monoids M, N is a *homomorphism* if $h(1) = 1$ and $h(x \cdot y) = h(x) \cdot h(y)$ for all $x, y \in M$. A language $L \subseteq \Sigma^*$ is *recognized* by a monoid M if there exists a homomorphism $h : \Sigma^* \rightarrow M$ from the free monoid Σ^* into a monoid M and a set $F \subseteq M$ such that $w \in L$ if and only if $h(w) \in F$ for all $w \in \Sigma^*$. It is well known that the

class of regular languages is exactly the class of languages recognized by finite monoids. For every language $L \subseteq \Sigma^*$ the *syntactic congruence* \equiv_L on Σ^* is defined by $u \equiv_L v$ if and only if for all $x, y \in \Sigma^*$: $xuy \in L$ if and only if $xvy \in L$. This is an equivalence relation and a congruence, i.e., $u \equiv_L v$ implies $xuy \equiv_L xvy$ for all $x \in \Sigma^*$. Thus, the set of congruence classes Σ^*/\equiv_L forms a monoid, which is called the *syntactic monoid* of L and is denoted by $M(L)$. A language L is regular if and only if $M(L)$ is a finite monoid. The function $h : \Sigma^* \rightarrow M(L)$ which maps a word u to its congruence class $[u]_{\equiv_L}$ is a surjective homomorphism, called the *syntactic homomorphism* of L , and it recognizes L . If L is regular, then $M(L)$ is the smallest monoid which recognizes L .

2.5 Graphs

In this paper all graphs are finite, directed and vertex-colored. For a graph Γ we denote by $V(\Gamma)$ and $E(\Gamma)$ the set of vertices and edges of Γ , respectively. Graphs may have loops, i.e. $E(\Gamma)$ is an arbitrary subset of $V(\Gamma) \times V(\Gamma)$. For graphs Γ and Δ , a *homomorphism* from Γ to Δ is a function $\varphi : V(\Gamma) \rightarrow V(\Delta)$ such that for all $v \in V(\Gamma)$ the vertices v and $\varphi(v)$ have the same color and $(u, v) \in E(\Gamma)$ implies $(\varphi(u), \varphi(v)) \in E(\Delta)$. We call a graph Γ *homomorphic* to Δ if there exists a homomorphism from Γ to Δ . For a subset $C \subseteq V(\Gamma)$ we denote by $\text{reach}_\Gamma(C)$ the subgraph of Γ which is induced by all nodes that are reachable from C . A graph Γ is *strongly connected* if for all $u \in V(\Gamma)$ we have $\text{reach}_\Gamma(\{u\}) = \Gamma$. A *strongly connected component*, briefly SCC, of Γ is an inclusion maximal subset $C \subseteq V(\Gamma)$ such that the subgraph induced by C is strongly connected. We identify C with this subgraph. The set of SCCs of a graph is partially ordered by $C_1 \preceq C_2$ if and only if a vertex in C_2 is reachable from a vertex in C_1 . An SCC of Γ is *trivial* if it consists of a single node v and $(v, v) \notin E(\Gamma)$, otherwise the SCC is called *non-trivial*. A graph is a *directed cycle* if it is strongly connected and every vertex has outdegree (and indegree) 1. Our characterizations of regular languages will refer to homomorphisms from certain graphs (that we define below) to directed cycles. Note that every monochromatic graph is homomorphic to a directed cycle of size one. In particular, a trivial SCC is homomorphic to a directed cycle.

For a monoid M and a subset A of M we denote by $\Gamma(M, A)$ the (unlabelled) *left Cayley graph* over the vertex set M with the edge set $\{(x, y) : y = a \cdot x \text{ for some } a \in A\}$. If the subset $A \subseteq M$ generates M , i.e. every element of M is a finite product over A , then $x \in M$ is reachable from $y \in M$ in $\Gamma(M, A)$ if and only if $x \leq_{\mathcal{L}} y$ in M , which is defined by

$$x \leq_{\mathcal{L}} y \iff \exists \ell \in M : x = \ell \cdot y. \quad (1)$$

The \mathcal{L} -equivalence is defined by $x \equiv_{\mathcal{L}} y$ if and only if $x \leq_{\mathcal{L}} y \leq_{\mathcal{L}} x$. For a subset $F \subseteq M$ we denote with $\Gamma(M, A, F)$ the graph $\Gamma(M, A)$, where in

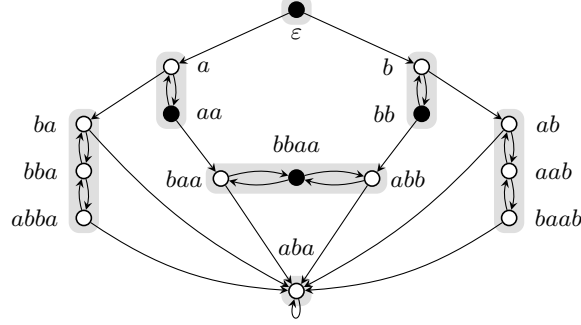


Figure 1: The left Cayley graph $\Gamma(M, A, F)$ from Example 2.1 for the language $(aa \mid bb)^*$. Vertices from F are black and SCCs are shaded.

addition all vertices from F (resp., $M \setminus F$) are colored with 1 (resp., 0).

Example 2.1. Consider the regular language $L = (aa \mid bb)^*$. Let $h : \{a, b\}^* \rightarrow M$ be the syntactic homomorphism of L into its syntactic monoid M with 15 elements. Figure 1 shows the left Cayley graph $\Gamma(M, A, F)$, where $A = \{h(a), h(b)\}$ and $F = h(L)$. Note that every SCC is homomorphic to a directed cycle.

3 Sliding window models

In the literature, one distinguishes two sliding window models: The *fixed-size model* and the *variable-size model*, see also [3] for a discussion of these models.

3.1 The fixed-size model

A *data stream* over Σ is a finite word $a_1 a_2 a_3 \cdots a_m$, where $a_i \in \Sigma$. The idea is that a data stream represents the sequence of data that is produced by some process.² At time t , the observer (or streaming algorithm) of this process can only see symbol a_t . Moreover, it is allowed to store and update in each step an internal data structure that we call the *storage content* below.

Formally, a *streaming algorithm* (in the fixed-size model) is a deterministic (not necessarily finite) automaton $\mathcal{A} = (S, \Sigma, \Phi, s_0, F)$ where

- $S \subseteq \{0, 1\}^*$ is a set of storage contents,
- Σ is the input alphabet,
- $s_0 \in S$ is the *initial storage content*,

²Some authors consider a data stream as an infinite sequence $a_1 a_2 a_3 \cdots$ of symbols. For our purpose this makes no essential difference.

- the *update mapping* $\Phi : S \times \Sigma \rightarrow S$ computes from the current storage content $s \in S$ and the current input symbol $a \in \Sigma$ the next storage content $\Phi(s, a)$, and
- $F \subseteq S$ is the set of *accepting storage contents*.

The goal of such a streaming algorithm is to determine at each time instant, whether the word consisting of the last n symbols from the input stream belong to a certain (fixed) language $L \subseteq \Sigma^*$. We define this word by $\text{last}_n(w) = v$ if $w = uv$ with $|v| = n$ and $\text{last}_n(w) = \square^{n-m}w$ if $|w| = m < n$, where $\square \in \Sigma$ is a distinguished symbol that fills the window initially. The length n is called the *window size* and $\text{last}_n(w)$ is called the *window content* at time $|w|$. The term “fixed-size model” is used here, since at each time $t \geq n$, the relevant part of the input stream has a fixed size n .

Formally, $L \subseteq \Sigma^*$ is *streamable in space $f(n)$ in the fixed-size model* if for every n there exists a streaming algorithm $\mathcal{A}_n = (S_n, \Sigma, \Phi_n, s_{n,0}, F_n)$ such that

- $L(\mathcal{A}_n) = \{w \in \Sigma^* : \text{last}_n(w) \in L\}$, and
- $|s| \leq f(n)$ for all $s \in S_n \subseteq \{0, 1\}^*$ (in particular, \mathcal{A}_n is a DFA).

Note that the fixed-size model is *non-uniform*: For every n we may have a separate streaming algorithm \mathcal{A}_n . This will not be crucial for our upper bounds, since our algorithms will work for all window sizes n (which is a parameter in the algorithms), with the only exception of Theorem 4.3 below. But working with a non-uniform model makes our lower bounds stronger.

3.2 The variable-size model

One can view the fixed-size model also as follows: A string (the window content) is dynamically modified as follows: At every time instant, a new symbol arrives and is appended on the right end of the string. Moreover, the first symbol of the string is removed. In contrast, in the *variable-size sliding window model* the arrival of new symbols and the expiration of old symbols is decoupled and can happen independently. This means that the window can grow and shrink. One can think of an adversary that executes a sequence of operations $\text{op}_1 \text{op}_2 \text{op}_3 \cdots \text{op}_m$, where every op_i is either a **pop**-operation or a **push**(a)-operation for a symbol $a \in \Sigma$. A **pop**-operation deletes the first symbol from the window; this corresponds to the situation where the first item in the window expires and falls out of the window (if the window is already empty it stays empty after a **pop**). A **push**(a)-operation appends the symbol a at the right end of the sliding window; this corresponds to the arrival of an a in the data stream. Formally, for a sequence $u \in \text{Op}(\Sigma)^*$ of operations, where $\text{Op}(\Sigma) = \{\text{pop}\} \cup \{\text{push}(a) : a \in \Sigma\}$, the window content $\text{window}(u) \in \Sigma^*$ is defined inductively as follows:

- $\text{window}(\varepsilon) = \varepsilon$,
- $\text{window}(u \text{ push}(a)) = \text{window}(u) a$.
- If $\text{window}(u) = \varepsilon$, then $\text{window}(u \text{ pop}) = \varepsilon$.
- If $\text{window}(u) = wa$ for $w \in \Sigma^*$ and $a \in \Sigma$, then $\text{window}(u \text{ pop}) = w$.

A streaming algorithm for the variable-size model is then a (not necessarily finite) deterministic automaton $\mathcal{A} = (S, \text{Op}(\Sigma), s_0, \Phi, F)$, where

- $S \subseteq \{0, 1\}^*$ is a set of storage contents,
- Σ is an alphabet,
- the *update mapping* $\Phi : S \times \text{Op}(\Sigma) \rightarrow S$ maps the current storage content $s \in \{0, 1\}^*$ and the current operation $\text{op} \in \text{Op}(\Sigma)$ to the next storage content $\Phi(s, \text{op})$,
- $F \subseteq S$ is the set of *accepting storage contents*, and
- $s_0 \in S$ is the *initial storage content*.

We say that the language $L \subseteq \Sigma^*$ is *streamable in space $f(n)$ in the variable-size model* if there exists a streaming algorithm \mathcal{A} as described above such that:

- (i) $L(\mathcal{A}) = \{u \in \text{Op}(\Sigma)^* : \text{window}(u) \in L\}$, and
- (ii) for all $u \in \text{Op}(\Sigma)^*$ with $n = \max\{|\text{window}(v)| : v \in \text{prefix}(u)\}$ we have $|\mathcal{A}(u)| \leq f(n)$.

Note the uniformity of this definition. There is a single update mapping Φ that has to work for every window size. We do not require that Φ is computable. This makes lower bounds with respect to the variable-size model stronger. On the other hand, in our upper bounds, the mapping Φ will be always efficiently computable with the only exception of Theorem 4.2 below.

In the conference version [17], we worked with a slightly stronger definition for the variable-size model. Instead of point (ii) in the above definition, we required that $|\mathcal{A}(u)| \leq f(|\text{window}(u)|)$. We think that condition (ii) is more natural. If at some time instant a certain space amount is used then the algorithm should be able to use this space also in the future, even if the window becomes shorter. This holds with our new definition (assuming that f is monotonic). Moreover, with our new definition we can show for every language L the existence of an optimal space bound among all monotonic functions; see Theorem 4.2 below. On the other hand, our space bounds for the variable-size model will always hold with respect to the stronger condition $|\mathcal{A}(u)| \leq f(|\text{window}(u)|)$.

The variable-size model captures various other streaming models that appeared in the literature. For instance, the standard model that was mentioned in the introduction corresponds to the case where no **pop**-operations are allowed. Another realistic model is the *time-stamp based model*, where the data items arrive at arbitrary time points (which are real numbers) and the sliding window contains all data values with an arrival time from the interval $[t - \tau, t]$, where t is the current time and τ is a fixed duration. Also the time-stamp based model can be simulated by the variable-size model, see [3] for details.

4 General space bounds

In this section, we prove several general results on the two sliding window models from Section 3. In contrast to our later results (for regular and context-free languages), these results apply to all languages.

4.1 Space-optimal streaming and language growth

For both streaming models, we will prove in this section the existence of optimal space bounds. Moreover, we will relate space to language growth.

First, we show that in the fixed-size as well as in the variable-size model there exist space-optimal streaming algorithms for any language. Let us start with the fixed-size model. A space-optimal streaming algorithm for $L \subseteq \Sigma^*$ and a given window size n can be constructed³ as follows. Recall that a streaming algorithm for L and a window size n is a deterministic (possibly infinite) automaton for

$$L_n = \{w \in \Sigma^* : \text{last}_n(w) \in L\}. \quad (2)$$

A particular DFA for L_n is

$$\mathcal{B}_\Sigma^n = (\Sigma^n, \Sigma, \delta_n, \square^n, L \cap \Sigma^n),$$

where $\delta_n(bv, a) = va$ for all $a, b \in \Sigma$ and $v \in \Sigma^{n-1}$ (recall that \square is the symbol that fills the window initially). This DFA is basically the n -dimensional De Bruijn graph over the alphabet Σ [12].

Let \mathcal{A}_L^n be the result of minimizing \mathcal{B}_L^n . Thus, \mathcal{A}_L^n is the minimal DFA for L_n and it has $|\Sigma^* / \sim_{L_n}|$ many states. A concrete example for \mathcal{B}_L^n and \mathcal{A}_L^n is shown in Figure 2 for window size $n = 2$ and the language $L = \{a, b\}^*b$ of all words that end with b . Define

$$\text{space}_L^{\text{fs}}(n) = \lfloor \log_2 |\Sigma^* / \sim_{L_n}| \rfloor. \quad (3)$$

The superscript “fs” stands for “fixed-size”.

³This construction is effective, if L is computable.

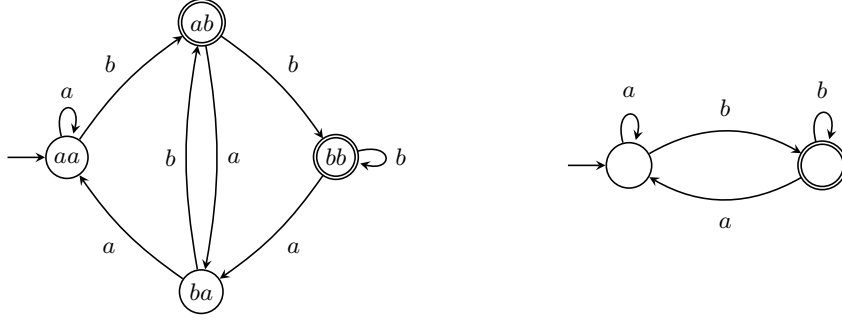


Figure 2: Let $L = \{a, b\}^*b$ and $\square = a$ (the initial window is aa). The DFA \mathcal{B}_L^2 is depicted on the left. The minimal DFA \mathcal{A}_L^2 is depicted on the right.

Theorem 4.1. *Let L be a language. The following holds:*

- L is streamable in space $\text{space}_L^{\text{fs}}(n)$ in the fixed-size model.
- If L is streamable in space $f(n)$ in the fixed-size model, then $f(n) \geq \text{space}_L^{\text{fs}}(n)$ for all $n \in \mathbb{N}$.

Proof. The states of the DFA \mathcal{A}_L^n can be encoded by bit strings of length at most $f_L(n)$. This shows that L is streamable in space $\text{space}_L^{\text{fs}}(n)$ in the fixed-size model. Now assume that L is streamable in space $f(n)$ in the fixed-size model. Fix a window size n and let \mathcal{A} be a streaming algorithm for L and window size n , whose state set is a subset of $\{0, 1\}^{\leq f(n)}$. Thus \mathcal{A} is a DFA for the language L_n with at most $2^{f(n)+1} - 1$ many states. Since \mathcal{A}_L^n is the minimal DFA for L_n , it has at most $2^{f(n)+1} - 1$ many states. This implies $\text{space}_L^{\text{fs}}(n) \leq \lfloor \log_2(2^{f(n)+1} - 1) \rfloor = f(n)$. \square

Theorem 4.1 says that the function $\text{space}_L^{\text{fs}}(n)$ is the optimal space bound for streaming L in the fixed-size model (where, as usual, n is the window size).

Next we construct a space-optimal streaming algorithm in the variable-size model. For a language $L \subseteq \Sigma^*$ (with Σ possibly infinite) we define the growth function $\gamma_L : \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\}$ as

$$\gamma_L(n) = |L \cap \Sigma^{\leq n}|.$$

Note that this definition makes also sense for an infinite alphabet Σ . In the following definitions we need this case. Define the mapping $\psi_L : \Sigma^* \rightarrow (\Sigma^*/\sim_L)^*$ by

$$\psi_L(a_1 \cdots a_n) = [a_1 \cdots a_n]_{\sim_L} [a_2 \cdots a_n]_{\sim_L} \cdots [a_n]_{\sim_L}.$$

Note that ψ_L is a length-preserving mapping from Σ^* to the free monoid generated by the set of all Myhill-Nerode equivalence classes (which is an

infinite set if L is not regular). Let $g_L = \gamma_{\psi_L(\Sigma^*)}$ be the growth function of the full image $\psi_L(\Sigma^*)$. If Σ is finite, then $g_L : \mathbb{N} \rightarrow \mathbb{N}$ indeed maps to \mathbb{N} . Clearly, $n + 1 \leq g_L(n) \leq |\Sigma|^n$. Let

$$\text{space}_L^{\text{vs}}(n) = \lfloor \log_2 g_L(n) \rfloor$$

(the superscript “vs” stands for “variable-size”). Thus, $\lfloor \log_2(n + 1) \rfloor \leq \text{space}_L^{\text{vs}}(n) \leq \lfloor (\log_2 |\Sigma|) \cdot n \rfloor$. Note that $\text{space}_L^{\text{vs}}(n)$ is monotonic.

Theorem 4.2. *If $\emptyset \subsetneq L \subsetneq \Sigma^*$, then the following holds:*

- L is streamable in the variable-size model in space $\text{space}_L^{\text{vs}}(n)$.
- If L is streamable in the variable-size model in space $f(n)$ where f is monotonic, then $f(n) \geq \text{space}_L^{\text{vs}}(n) \geq \lfloor \log_2(n + 1) \rfloor$ for all $n \geq 0$.

Proof. First we notice that given $\psi_L(a_1 \cdots a_n)$ one can compute $\psi_L(a_2 \cdots a_n)$ by removing the first \sim_L -class $[a_1 \cdots a_n]_{\sim_L}$. Moreover, for every $a \in \Sigma$ one can compute $\psi_L(a_1 \cdots a_n a) = [a_1 \cdots a_n a]_{\sim_L} [a_2 \cdots a_n a]_{\sim_L} \cdots [a_n a]_{\sim_L} [a]_{\sim_L}$ from $\psi_L(a_1 \cdots a_n)$, since \sim_L is a right-congruence (which means that $[w]_{\sim_L}$ determines $[wa]_{\sim_L}$). Clearly, the first \sim_L -class in $\psi_L(a_1 \cdots a_n)$ also determines whether $a_1 \cdots a_n \in L$. These remarks define a streaming algorithm for L in the variable-size model. However, instead of storing $\psi_L(a_1 \cdots a_n)$ explicitly, one can store a bit string of length $O(\log g_L(n))$ that encodes the position of $\psi_L(a_1 \cdots a_n)$ in the length-lexicographic enumeration of all words from $\psi_L(\Sigma^*)$.

Conversely, consider a streaming algorithm \mathcal{A} for L that works in space $f(n)$ for a monotonic function f . We have to show that $f(n) \geq \lfloor \log_2 g_L(n) \rfloor$. Consider an input word $x = a_1 a_2 \cdots a_m \in \Sigma^{\leq n}$. Let

$$s(x) = \mathcal{A}(\text{push}(a_1)\text{push}(a_2) \cdots \text{push}(a_m)) \quad (4)$$

(the storage content of the algorithm after processing the sequence of push-operations $\text{push}(a_1)\text{push}(a_2) \cdots \text{push}(a_m)$). Note that $|s(x)| \leq f(m) \leq f(n)$ by monotonicity of f .

We first show that $s(x)$ determines $m = |x|$. Assume that $\varepsilon \in L$ (the case that $\varepsilon \notin L$ is analog), and let $y \notin L$ where $|y|$ is chosen minimally. From $s(x)$ we can compute $s(xy)$ by performing the operations $\text{push}(b_1)\text{push}(b_2) \cdots \text{push}(b_k)$, where $y = b_1 b_2 \cdots b_k$. Then we perform an infinite sequence of **pop**-operations. Of course this is not possible effectively, but this is not problem. All we want to show is that $s(x)$ uniquely determines m (but we do not need to compute m effectively from $s(x)$). Let $c_0, c_1, c_2, \dots \in \{0, 1\}$ be the infinite bit sequence where c_i indicates whether the window content belongs to L after performing i **pop**-operations starting from $s(xy)$. Since $c_{|x|} = 0$ and $c_i = 1$ for all $i > m$, one can retrieve m from this infinite bit sequence.

We now show that $s(x)$ uniquely determines $\psi_L(a_1 \cdots a_m)$. By the above argument, we know that $s(x)$ determines the window size m . Furthermore we can determine the equivalence class $[a_k \cdots a_m]_{\sim_L}$ from $s(x)$ for every $1 \leq k \leq m$ as follows: Let $\{x_i : i \in I\}$ be a (possibly infinite) set of representatives of the \sim_L -classes. For $i, j \in I$ with $i \neq j$ let $y_{i,j} \in \Sigma^*$ be such that $x_i y_{i,j} \in L$ if and only if $x_j y_{i,j} \notin L$. To determine $[a_k \cdots a_m]_{\sim_L}$ from $s(x)$ we perform $k - 1$ **pop**-operations. Then, the window content is $a_k \cdots a_m$. For all $i, j \in I$ with $i \neq j$ let $s_{i,j}$ be the storage content after pushing $y_{i,j}$, which yields the window content $a_k \cdots a_m y_{i,j}$. The storage content $s_{i,j}$ determines whether $a_k \cdots a_m y_{i,j} \in L$ or not.

There is exactly one $l \in I$ such that for all $y_{i,j}$: $a_k \cdots a_m y_{i,j} \in L$ if and only if $x_l y_{i,j} \in L$. Moreover, this is the unique l with $x_l \sim_L a_k \cdots a_m$. This shows that the \sim_L -class of $a_k \cdots a_m$ is determined by the storage contents $s_{i,j}$ ($i \neq j$) and hence by $s(x)$.

To sum up, we have shown that every value $\psi_L(x)$ for $x \in \Sigma^{\leq n}$ can be encoded by a bit string of length at most $f(n)$ (namely $s(x)$). Since there are $g_L(n)$ such values, it follows that $2^{f(n)+1} - 1 \geq g_L(n)$, which implies $f(n) \geq \lfloor \log_2 g_L(n) \rfloor$. \square

If L is streamable in the variable-size model in space $f(n)$, where f is not monotonic but unbounded, then we can define a new monotonic function $f'(n) = \max\{f(i) : 0 \leq i \leq n\}$. Clearly, L is also streamable in space $f'(n)$, which by Theorem 4.2 implies that $f'(n) \geq \text{space}_L^{\text{vs}}(n)$ for all $n \geq 0$. Hence, there exist infinitely many n with $f(n) \geq \text{space}_L^{\text{vs}}(n)$.

Also note that $\text{space}_L^{\text{fs}}(n) \leq \text{space}_L^{\text{vs}}(n)$ for all $n \geq 0$. We simply run the variable-size algorithm from the first part of the proof of Theorem 4.2 on all operation sequences from $\{\text{push}(a) \mid a \in \Sigma\}^n \{\text{pop push}(a) \mid a \in \Sigma\}^*$ in order to get a fixed-size streaming algorithm for window size n that uses space $\text{space}_L^{\text{vs}}(n)$. Hence $\text{space}_L^{\text{fs}}(n) \leq \text{space}_L^{\text{vs}}(n)$ follows from Theorem 4.1.

The following theorem shows a connection between language growth and the space complexity in the fixed-size model:

Theorem 4.3. *For every language L , $\text{space}_L^{\text{fs}}(n) \in O(\log \gamma_L(n) + \log n)$.*

Proof. Fix the window size n and let w_1, \dots, w_m with $m = \gamma_L(n)$ be an arbitrary enumeration of $L \cap \Sigma^n$. Assume that w is the current window content. The algorithm stores (i) the binary encoding of the smallest position $1 \leq i \leq n + 1$ such that $w[i :]$ is a prefix of a word w_j , $1 \leq j \leq m$ (using $\log n$ many bits) and (ii) the binary encoding of the number j (using $\log \gamma_L(n)$ many bits). Of course, there may exist several words w_j having $w[i :]$ as a prefix; in this case the concrete choice of w_j does not matter. This information clearly suffices to check whether the window content belongs to L . Moreover, we can update the information: If a is the next symbol from the stream, then we distinguish the following cases:

- If $i > 1$ and $w[i:]a$ is a prefix of a word from $L \cap \Sigma^n$, say $w_{j'}$, $1 \leq j' \leq m$, then we replace i, j by $i - 1, j'$.
- Otherwise let $k \geq 2$ be minimal such that $(w[i:]a)[k:]$ is a prefix of a word from $L \cap \Sigma^n$, say $w_{j'}$, $1 \leq j' \leq m$. We replace i, j by $i - 2 + k, j'$.

The correctness of this algorithm is straightforward. \square

Note the non-uniformity of the above algorithm. This is of course necessary to get the result since there are non-recursive languages of constant growth. The following example shows that Theorem 4.3 is not true in the variable-size model.

Example 4.4. Let $L = \{awb^{2^{|w|}} : w \in \{a, b\}^*\}$. Note that each word in L has length $2^k + k + 1$ for some $k \in \mathbb{N}$. Now consider $n = 2^k + k + 1$ for some $k \in \mathbb{N}$. We have $|L \cap \Sigma^n| = 2^k$, which together with $k \in \Theta(\log n)$ shows that L has polynomial growth.

We prove $\text{space}_L^{\text{vs}}(n) \notin o(n)$ which implies that L is not streamable in space $O(\log \gamma_L(n) + \log n)$ in the variable-size model. Towards a contradiction, assume $\text{space}_L^{\text{vs}}(n) = f(n) \in o(n)$ and let the deterministic automaton \mathcal{A} be a corresponding streaming algorithm, which realizes this space bound. Consider the set of sequences of operations $M_n = \{\text{push}(a), \text{push}(b)\}^n$. For each n , we have $|\mathcal{A}(u)| \leq f(n)$ for all $u \in M_n$. Since $|M_n| = 2^n$, it follows that for n large enough, there exist $u, u' \in M_n$ with $u \neq u'$ such that $\mathcal{A}(u) = \mathcal{A}(u')$. Let i be a position ($1 \leq i \leq n$) such that $u[i] \neq u'[i]$. Without loss of generality, we assume $u[i] = \text{push}(a)$ and $u'[i] = \text{push}(b)$. Now consider the words $x = \text{window}(u \text{ pop}^{i-1} \text{ push}(b)^{2^{n-i}})$ and $x' = \text{window}(u' \text{ pop}^{i-1} \text{ push}(b)^{2^{n-i}})$. We have $x = awb^{2^{|w|}}$ and $x' = bw'b^{2^{|w'|}}$ for some $w, w' \in \{a, b\}^{n-i}$. It follows that $x \in L$ and $x' \notin L$, which contradicts $\mathcal{A}(u) = \mathcal{A}(u')$ and thus $\text{space}_L^{\text{vs}}(n) \notin o(n)$.

4.2 Suffix testable languages

In this section we establish a connection between so-called suffix testable languages and the space complexity in the fixed-size model. A language $L \subseteq \Sigma^*$ is called *k-suffix testable* if for all $s, s' \in \Sigma^*$ and $t \in \Sigma^k$ we have

$$st \in L \iff s't \in L.$$

Equivalently, L is a finite Boolean combination of languages of the form Σ^*w where $w \in \Sigma^k$. Recall the definition of $L_n = \{w \in \Sigma^* : \text{last}_n(w) \in L\}$ from (2), which is the language accepted by a streaming algorithm for window size n . We will show that L_n is $2^{\text{space}_L^{\text{fs}}(n)+1}$ -suffix testable. To do so, the following definitions are useful, which are also studied in [18].

For two languages $K, L \subseteq \Sigma^*$, we denote by $K \triangle L = (K \setminus L) \cup (L \setminus K)$ the symmetric difference of K and L . We define the distance $d(K, L)$ by

$$d(K, L) = \begin{cases} \sup\{|u| : u \in K \triangle L\} + 1, & \text{if } K \neq L, \\ 0, & \text{if } K = L. \end{cases}$$

Notice that $d(K, L) < \infty$ if and only if $K \triangle L$ is finite. For a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and a state $p \in Q$, we define $\mathcal{A}_p = (Q, \Sigma, \delta, p, F)$. For two states $p, q \in Q$, we define the distance $d(p, q) = d(L(\mathcal{A}_p), L(\mathcal{A}_q))$. We have $d(p, q) = \max_{a \in \Sigma} d(\delta(p, a), \delta(q, a)) + 1$ for all $a \in \Sigma$, and if $d(p, q) < \infty$ then $d(p, q) \leq |Q|$, see [18, Lemma 1].

Lemma 4.5. *Let L be a regular language and $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ its minimal DFA. Then the following hold:*

- (i) *L is k -suffix testable if and only if $\mathcal{A}(st) = \mathcal{A}(s't)$ for all $s, s' \in \Sigma^*$, $t \in \Sigma^k$.*
- (ii) *L_n is k -suffix testable for all n if and only if $\mathcal{A}(st) = \mathcal{A}(s't)$ for all $s, s' \in \Sigma^*$, $t \in \Sigma^k$ with $|s| = |s'|$.*
- (iii) *If L is k -suffix testable for some k , then L is $|Q|$ -suffix testable.*

Proof. For the “only if”-direction of (i) assume that L is k -suffix testable and let $s, s', u \in \Sigma^*$, $t \in \Sigma^k$ arbitrary. Since stu and $s'tu$ share a suffix of length $|tu| \geq k$ we have $\mathcal{A}(stu) \in F$ if and only if $\mathcal{A}(s'tu) \in F$. Hence the states $\mathcal{A}(st)$ and $\mathcal{A}(s't)$ are identical because \mathcal{A} is a minimal DFA. For “if”-direction let $s, s' \in \Sigma^*$ be arbitrary. The assumption implies that $\mathcal{A}(st) = \mathcal{A}(s't)$ for every word t of length at least k . In particular $st \in L$ if and only if $s't \in L$.

The proof of (ii) is similar: For the “only if”-direction of (i) assume that L_n is k -suffix testable for all n and let $s, s', u \in \Sigma^*$, $t \in \Sigma^k$ where $|s| = |s'|$. Let $n = |stu|$. Since stu and $s'tu$ share a suffix of length $|tu| \geq k$ and L_n is k -suffix testable we have $stu \in L_n$ if and only if $s'tu \in L_n$. Since $\text{last}_n(stu) = stu$ and $\text{last}_n(s'tu) = s'tu$ we have $\mathcal{A}(stu) \in F$ if and only if $\mathcal{A}(s'tu) \in F$. Hence the states $\mathcal{A}(st)$ and $\mathcal{A}(s't)$ are identical because \mathcal{A} is a minimal DFA. For the “if”-direction assume that $\mathcal{A}(st) = \mathcal{A}(s't)$ for all $s, s' \in \Sigma^*$, $t \in \Sigma^k$ with $|s| = |s'|$. We have to show that $\text{last}_n(ut) \in L$ if and only if $\text{last}_n(u't) \in L$ for all $u, u' \in \Sigma^*$ and $t \in \Sigma^k$. If $n < k$, then this is clear because $\text{last}_n(ut) = \text{last}_n(t) = \text{last}_n(u't)$. If $n \geq k$, then there exist s, s' with $|s| = |s'|$ and $\text{last}_n(ut) = st$ and $\text{last}_n(u't) = s't$. Hence, we have $\mathcal{A}(\text{last}_n(ut)) = \mathcal{A}(\text{last}_n(u't))$ and both words are either accepted or rejected by \mathcal{A} .

For (iii) assume that L is k -suffix testable, i.e., $\mathcal{A}(st) = \mathcal{A}(s't)$ for all $s, s' \in \Sigma^*$ and $t \in \Sigma^k$. In particular $d(\mathcal{A}(s), \mathcal{A}(s')) < \infty$ for all $s, s' \in \Sigma^*$ and we have $d(\mathcal{A}(s), \mathcal{A}(s')) \leq |Q|$ by the remark above. Hence, for any $s, s' \in \Sigma^*$,

$t \in \Sigma^{|Q|}$ we have $d(\mathcal{A}(st), \mathcal{A}(s't)) = 0$, i.e., $st \in L$ if and only if $s't \in L$. Therefore L is indeed $|Q|$ -suffix testable. \square

Theorem 4.6. *For every language $L \subseteq \Sigma^*$, L_n is $2^{\text{space}_L^{\text{fs}}(n)+1}$ -suffix testable.*

Proof. Recall the definition of the DFA $\mathcal{A}_L^n = (Q_n, \Sigma, \delta_n, q_n, F_n)$, which is the smallest DFA for the language $L_n = \{w \in \Sigma^* : \text{last}_n(w) \in L\}$ from (2). For every n , the language L_n is n -suffix testable by its definition. Point (iii) of Lemma 4.5 implies that every language L_n is $|Q_n|$ -suffix testable. Together with $|Q_n| \leq 2^{\text{space}_L^{\text{fs}}(n)+1}$ (see equation (3)) this proves the theorem. \square

Corollary 4.7. *Let $L \subseteq \Sigma^*$. We have $\text{space}_L^{\text{fs}}(n) \in O(1)$ if and only if there exists a constant k such that for all n the language L_n is k -suffix testable.*

Proof. The “only if”-direction follows directly from Theorem 4.6. On the other hand, if there exists a constant k such that L_n is k -suffix testable for all n , then L is streamable in the fixed-size model in constant space since we only need to store and update the last k symbols of the current window content. \square

Corollary 4.8. *Let $L \subseteq \Sigma^*$ be regular and $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be its minimal DFA. We have $\text{space}_L^{\text{fs}}(n) \in O(1)$ if and only if $\mathcal{A}(st) = \mathcal{A}(s't)$ for all $s, s' \in \Sigma^*$, $t \in \Sigma^{|Q|}$ with $|s| = |s'|$.*

Proof. This is a direct corollary of Corollary 4.7 and Lemma 4.5. \square

5 Streaming algorithms for regular languages

In this section, we will prove a trichotomy for the optimal space needed to query a regular language in the sliding window model. Let $L \subseteq \Sigma^*$ be a regular language. Let $M = M(L)$ be the syntactic monoid of L and $h : \Sigma^* \rightarrow M$ be the syntactic homomorphism. Let $F = h(L)$, hence $L = h^{-1}(F)$. We simply write Γ for the two-colored left Cayley graph $\Gamma(M, A, F)$ where $A = h(\Sigma)$. Recall that Γ is a finite directed graph, possibly with loops. For the rest of Section 5 we fix Σ , L , M , h , A , and Γ . It is important for our results that L is fixed, and not part of the input. This implies that the monoid M and the graph Γ can be hard-wired into our algorithms.

We partition the set of all regular languages over Σ into three classes \mathcal{C}_1 , \mathcal{C}_2 , \mathcal{C}_3 , where

- $L \in \mathcal{C}_1$ if and only if for every non-trivial SCC C of Γ the subgraph $\text{reach}_\Gamma(C)$ is homomorphic to a directed cycle,
- $L \in \mathcal{C}_2$ if and only if $L \notin \mathcal{C}_1$ and every SCC of Γ is homomorphic to a directed cycle,
- $L \in \mathcal{C}_3$ if and only if $L \notin (\mathcal{C}_1 \cup \mathcal{C}_2)$.

	constant space	logarithmic space	linear space
fixed-size model	\mathcal{C}_1	\mathcal{C}_2	\mathcal{C}_3
variable-size model	$\{\emptyset, \Sigma^*\}$	$(\mathcal{C}_1 \cup \mathcal{C}_2) \setminus \{\emptyset, \Sigma^*\}$	\mathcal{C}_3

Table 1: The trichotomy results for querying regular languages in the sliding window model.

For instance, the language $(aa \mid bb)^*$ from Example 2.1 belongs to \mathcal{C}_2 . Other examples for languages in $\mathcal{C}_1 \cup \mathcal{C}_2$ are *left open* languages, i.e. languages of the form Σ^*L where L is a regular language over Σ . Examples for languages in \mathcal{C}_1 are languages of the form Σ^*w for $w \in \Sigma^*$.

For the fixed-size model we will show that

- $\text{space}_L^{\text{fs}}(n) \in O(1)$ for languages $L \in \mathcal{C}_1$,
- $\text{space}_L^{\text{fs}}(n) \in O(\log n) \setminus o(\log n)$ for languages $L \in \mathcal{C}_2$,
- $\text{space}_L^{\text{fs}}(n) \in O(n) \setminus o(n)$ for languages $L \in \mathcal{C}_3$.

For the variable-size model we will show that

- $\text{space}_L^{\text{vs}}(n) \in O(1)$ for the trivial languages \emptyset and Σ^* ,
- $\text{space}_L^{\text{vs}}(n) \in \Theta(\log n)$ for languages $L \in (\mathcal{C}_1 \cup \mathcal{C}_2) \setminus \{\emptyset, \Sigma^*\}$,
- $\text{space}_L^{\text{vs}}(n) \in \Theta(n)$ for languages in $L \in \mathcal{C}_3$.

Note that by Corollary 4.7 the regular languages L with $\text{space}_L^{\text{fs}}(n) \in O(1)$ are the regular languages that are k -suffix testable for some constant k . To show that $\text{space}_L^{\text{fs}}(n) \in o(\log n)$ implies $\text{space}_L^{\text{fs}}(n) \in O(1)$ for a regular L , the above definition of \mathcal{C}_1 turns out to be useful.

Example 5.1. Let $L_1 = \{a, b\}^*a$ be the set of all words that end with an a . Obviously, L_1 is streamable in constant space in the fixed-size model: At each time t , the algorithm only needs to store whether the current symbol a_t is an a or not. This can be done using a single bit. Similarly, for every finite word $w \in \{a, b\}^*$ the language $\{a, b\}^*w$ is streamable in constant space in the fixed-size model: The algorithm has to store the last $|w|$ symbols from the stream. Note that this argument fails for the variable-size model: In fact, L_1 is not streamable in constant space in the variable-size model; this follows from Theorem 4.2.

Example 5.2. Let $L_2 = \{a, b\}^*a\{a, b\}^*$ be the set of all words that contain an a . This language is streamable in space $O(\log n)$ in the variable-size model. The algorithm stores (i) the current window size n (using $O(\log n)$ bits), and (ii) the position p of the right-most a in the window (using $O(\log n)$

bits). We set p to 0 if the window contains no a . This information can be easily updated: For a **pop**-operation, the algorithm sets $n := \max\{0, n-1\}$ and $p := \max\{0, p-1\}$. For a **push**(a)-operation, the algorithm sets $n := n+1$ and $p := n$. Finally, for a **push**(b)-operation only n is incremented.

On the other hand, L_2 is not streamable in space $o(\log n)$ in the fixed-size model: If L_2 would be streamable in space $o(\log n)$ then one could represent every number $1 \leq i \leq n$ by a bit string of length $o(\log n)$, namely by the $o(\log n)$ -size storage content $s(i)$ obtained after reading the word $b^{i-1}ab^{n-i}$, where n is the window size. To recover i from $s(i)$ one runs the streaming algorithm, starting with storage content $s(i)$, and reads a b in every step. The smallest number of b 's after which a membership query for L_2 is answered negatively is i .

Example 5.3. Let $L_3 = a\{a, b\}^*$ be the set of all words that start with an a . An argument similar to Example 5.2 shows that L_3 is not streamable in space $o(n)$ in the fixed-size model. More precisely, if L_3 would be streamable in space $o(n)$ in the fixed-size model, then one could represent every word $w \in \{a, b\}^n$ by a bit string of length $o(n)$, namely by the $o(n)$ -size storage content $s(w)$ obtained after reading the word w , where $n = |w|$ is the window size. To recover w from $s(w)$ one runs the streaming algorithm, starting with storage content $s(w)$. The query result after reading $i-1$ further input symbols yields the i -th symbol of w : A positive (resp., negative) query answer yields an a (resp., b).

5.1 Upper bounds

In this section we will prove two upper bounds on the space for querying regular languages in the sliding window model. First, we show that every language in $\mathcal{C}_1 \cup \mathcal{C}_2$ is streamable in logarithmic space in both streaming models.

Theorem 5.4. *If every SCC of Γ is homomorphic to a directed cycle, then $\text{space}_L^{\text{vs}}(n) \in O(\log n)$ and hence also $\text{space}_L^{\text{fs}}(n) \in O(\log n)$.*

Proof. Since the fixed-size model can be simulated by the variable-size model, it suffices to present an algorithm for the variable-size model.

Let $w \in \Sigma^*$ be a word of length n and $\text{Suf}(w)$ be the set of suffixes of w , which includes the empty word and w itself. Define the preorder \preceq on $\text{Suf}(w)$ by $u \preceq v$ if and only if $h(u) \leq_{\mathcal{L}} h(v)$, where $\leq_{\mathcal{L}}$ is defined in (1). This is in fact a total preorder: If $v \in \text{Suf}(w)$ is a suffix of $u \in \text{Suf}(w)$ then $u \preceq v$. But note that we may have $u \preceq v \preceq u$ for two different suffixes of w . The word w is a smallest element w.r.t. \preceq . The induced equivalence relation \equiv is defined by $u \equiv v$ if and only if $u \preceq v \preceq u$. Clearly, $u \equiv v$ if and only if $h(u) \equiv_{\mathcal{L}} h(v)$. As usual, denote with $\text{Suf}(w)/\equiv$ the set of equivalence classes of \equiv . Note that $|\text{Suf}(w)/\equiv|$ is bounded by a constant

which only depends on the monoid M and not on the window size n . One can identify the elements of $\text{Suf}(w)/\equiv$ with intervals on the set of positions of w . Hence we can represent $(\text{Suf}(w), \preceq)$ by storing a constant number of interval endpoints using $O(\log n)$ bits. Our streaming algorithm (for window size n) stores the following data:

- the total preorder $(\text{Suf}(w), \preceq)$, using $O(\log n)$ bits,
- the function $f : \text{Suf}(w)/\equiv \rightarrow M$ defined by $f(D) = h(v)$ where v is the shortest suffix in the equivalence class D , using $O(1)$ bits.

We describe these data conveniently by a sequence

$$p_0, m_1, p_1, m_2, p_2, \dots, m_{k-1}, p_{k-1}, m_k, p_k \quad (5)$$

such that the following holds:

- $1 \leq k \leq |M|$,
- $0 = p_0 < p_1 < \dots < p_{k-1} < p_k = n + 1$,
- $m_1, \dots, m_k \in M$ and m_k is the unit element of M .

The meaning of this sequence is the following: The equivalence classes of \equiv are the sets $D_i = \{w[p : n] : p_{i-1} < p \leq p_i\}$ for $1 \leq i \leq k$ (the class D_k contains the empty suffix for $p = p_k = n + 1$). The monoid element m_i is $h(w[p_i : n])$ for $1 \leq i \leq k$ (hence, $m_k = 1$ is the unit element). Thus, $m_i = h(v)$ where v is the shortest suffix in its equivalence class D_i .

On the sequence (5) we can now perform the desired queries: In order to test whether $w \in L$, one has to check whether $h(w) \in F$. For this we consider the monoid element m_1 . Note that $m_1 \equiv_{\mathcal{L}} h(w)$. Hence, the vertices $h(w)$ and $m_1 = h(w[p_1 : n])$ belong to the same SCC C of Γ . Note that $h(w) = h(w[1 : p_1 - 1])m_1$. We cannot store this word $w[1 : p_1 - 1]$, in fact we do not even store its image under h . But, by assumption, the SCC C of Γ that contains $h(w)$ and m_1 has a homomorphism φ onto a directed cycle Θ . Thus, we can compute $\varphi(h(w))$ by traversing the cycle from $\varphi(m_1)$ for $p_1 - 1$ steps (the homomorphic image of Γ under φ is hard-wired into the algorithm). The color of $\varphi(h(w))$ in Θ then indicates whether $h(w) \in F$ (i.e. $w \in L$) or $h(w) \notin F$ (i.e. $w \notin L$).

For a **pop**-operation on w and $p_1 > 1$, the algorithm updates the sequence (5) to

$$p_0, m_1, p_1 - 1, m_2, p_2 - 1, \dots, m_{k-1}, p_{k-1} - 1, m_k, p_k - 1.$$

Otherwise, if $p_1 = 1$ then the algorithm updates the sequence (5) to

$$p_0, m_2, p_2 - 1, \dots, m_{k-1}, p_{k-1} - 1, m_k, p_k - 1.$$

Finally, let us consider a $\text{push}(a)$ -operation on w . Note that $\equiv_{\mathcal{L}}$ is a right congruence, i.e. $x \equiv_{\mathcal{L}} y$ implies $xz \equiv_{\mathcal{L}} yz$ for all $x, y, z \in M$. This means that our interval-representation of $(\text{Suf}(wa), \preceq)$ can be obtained from the interval-representation of $(\text{Suf}(w), \preceq)$ by possibly merging successive intervals. In order to detect, which intervals have to be merged, note that for all $u, v \in \text{Suf}(w)$ we have

$$ua \equiv va \iff h(u)h(a) \equiv_{\mathcal{L}} h(v)h(a) \iff f([u]_{\equiv})h(a) \equiv_{\mathcal{L}} f([v]_{\equiv})h(a),$$

because $h(u) \equiv_{\mathcal{L}} f([u]_{\equiv})$ and $h(v) \equiv_{\mathcal{L}} f([v]_{\equiv})$, and the fact that $\equiv_{\mathcal{L}}$ is a right congruence. Using this, we can detect whether two successive intervals that represent the classes $[u]_{\equiv}$ and $[v]_{\equiv}$ have to be merged into a single interval. Formally, we process the sequence (5) as follows: We walk over the sequence from left to right. For every $1 \leq i \leq k-1$ we check whether $m_i h(a) \equiv_{\mathcal{L}} m_{i+1} h(a)$. If this is true, then we remove m_i, p_i from the sequence, otherwise we replace m_i, p_i by $m_i h(a), p_i$. Then we continue with $i+1$ (if $i < k-1$). Finally, we check whether $h(a) \equiv_{\mathcal{L}} 1$. If this holds, then we replace $m_k, p_k = 1, n+1$ by $1, n+2$, otherwise we replace $1, n+1$ by $h(a), n+1, 1, n+2$. \square

Next we show that languages in \mathcal{C}_1 are streamable in constant space in the fixed-size model.

Theorem 5.5. *If $\text{reach}_{\Gamma}(C)$ is homomorphic to a directed cycle for every non-trivial SCC C of Γ , then $\text{space}_L^{\text{fs}}(n) \in O(1)$.*

Proof. Observe that every path in Γ of length at least $c := |V(\Gamma)|$ (a constant) contains a vertex in a non-trivial SCC C and therefore ends in $\text{reach}_{\Gamma}(C)$. Fix a window size n . If $n < c$, we store the window content explicitly and can test whether $w \in L$, e.g. using an automaton for L . Now assume $n \geq c$. For a window content $w \in \Sigma^*$ we explicitly store the suffix v of length c . Clearly this suffix can be updated when a new symbol arrives in the window. Also v suffices to test whether $w \in L$. We compute $h(v)$ and a non-trivial SCC C such that $h(v)$ is contained in $\text{reach}_{\Gamma}(C)$. Let $\varphi : \text{reach}_{\Gamma}(C) \rightarrow \Theta$ be the homomorphism into a directed cycle Θ . Then we compute $\varphi(h(w))$ by traversing Θ starting from the vertex $\varphi(h(v))$ for $n-c$ steps. The color of $\varphi(h(w))$ determines whether $w \in L$. \square

As in most previous work on the sliding window model, our focus is on the space requirements of query algorithms. But it is also interesting to note that in Theorem 5.4 and 5.5 we can achieve constant time for all update and query operations on the RAM model with register length $O(\log n)$. Let us show this for Theorem 5.4 first. Recall that the sequence (5) that we manipulate in the proof of Theorem 5.4 has constant length. For a pop - or $\text{push}(a)$ -operation, the manipulation of (5) only needs constant time. To see this, note that the numbers p_i in (5) are only incremented or decremented and that all operations in the monoid M need constant time since M is

fixed. Finally, for a membership query we traverse the cycle Θ starting from $\varphi(m_1)$ for $p_1 - 1$ steps. To do this in constant time, we store also the numbers $(p_i - 1) \bmod \ell(\Theta)$, where $\ell(\Theta)$ is the length of the cycle. We have to maintain these remainders for all (constantly many) cycle lengths $\ell(\Theta_1), \dots, \ell(\Theta_c)$, where $\Theta_1, \dots, \Theta_c$ are the cycles to which the SCCs of Γ are homomorphic. For Theorem 5.5 it suffices to traverse Θ for $(n - c) \bmod \ell(\Theta)$ steps.

5.2 Lower bounds

In this section, we prove matching lower bounds for the upper bounds from the previous section. Recall from Theorem 4.2 constant space in the variable-size model makes it impossible to query any non-trivial language. In this section we prove two lower bounds for regular languages. For this we need the following simple graph theoretic lemma:

Lemma 5.6. *Let Γ be a finite directed vertex-colored graph (possibly with loops) and let s be a vertex from which all vertices of Γ are reachable. Assume that all vertices have outdegree ≥ 1 and s has indegree ≥ 1 . If Γ is not homomorphic to a directed cycle, then there exist paths π_0, π_1 of the same length from s to vertices s_0, s_1 which have distinct colors.*

Proof. Let V_n be the set of vertices which are reachable from s via a path of length n for $n \geq 0$. The union $\bigcup_{n \geq 0} V_n$ is the set of vertices reachable from s , which by assumption is $V(\Gamma)$. Towards a contradiction assume that every set V_n is monochromatic. Let \approx be the transitive-reflexive closure of the binary relation R on $V(\Gamma)$ defined by $R = \bigcup_{n \geq 0} V_n \times V_n$. Then, every equivalence class of \approx is monochromatic. Hence, we can construct the quotient graph $\Gamma / \approx = (\{[v]_\approx : v \in V(\Gamma)\}, \{([u]_\approx, [v]_\approx) : (u, v) \in E(\Gamma)\})$. Moreover, the equivalence class $[u]_\approx$ has the same color as all its elements. Clearly, Γ is homomorphic to Γ / \approx .

We claim that every vertex in Γ / \approx has out-degree 1: Since every vertex in Γ has outdegree ≥ 1 , the same holds for Γ / \approx . Moreover, if a vertex v is contained in some set V_n , then all successors of v are contained in V_{n+1} . This implies that R respects the successor relation, i.e. whenever $(u, v) \in R$ and $(u, u'), (v, v') \in E(\Gamma)$, then also $(u', v') \in R$. Hence, also \approx respects the successor relation. This proves that every vertex in Γ / \approx has out-degree 1. Finally, each node has an incoming edge since s has an incoming edge and all other nodes are reachable from s . It follows that Γ / \approx must in fact be a directed cycle. \square

Now we show that languages in \mathcal{C}_3 are not streamable in space $o(n)$ in the fixed-size model.

Theorem 5.7. *If some SCC C of Γ is not homomorphic to a directed cycle, then $\text{space}_L^{\text{fs}}(n) \notin o(n)$ and $\text{space}_L^{\text{vs}}(n) \in \Omega(n)$.*

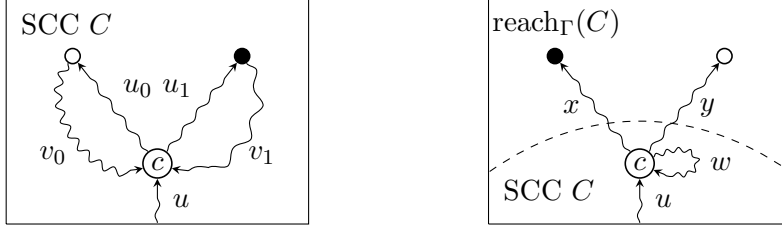


Figure 3: The origin of the words used in the proofs of Theorem 5.7 (left) and Theorem 5.8 (right).

Proof. We apply Lemma 5.6 with an arbitrary node $c \in C$ to the subgraph of Γ induced by C . Therefore, there exist paths π_0 and π_1 of the same length k from c to nodes $c_0, c_1 \in C$, which are colored differently, say $c_0 \notin F$ and $c_1 \in F$. Let $u_0, u_1 \in \Sigma^k$ be words representing the paths π_0, π_1 and let $u \in \Sigma^*$ such that $h(u) = c$, which exists since h is surjective. Since C is strongly connected, there also exist paths π'_0 and π'_1 from c_0 and c_1 , respectively, back to c . This results in words $v_0, v_1 \in \Sigma^+$ such that $h(v_0 u_0 u) = h(u) = h(v_1 u_1 u)$, $h(u_0 u) \notin F$, $h(u_1 u) \in F$. The situation is shown in Figure 3 on the left. Choose numbers $p, q > 0$ such that $z_0 = (v_0 u_0)^p$ and $z_1 = (v_1 u_1)^q$ have the same length. We get $h(z_0 u) = h(z_1 u) = h(u)$. Let x_0, x_1 such that $z_0 = x_0 u_0$ and $z_1 = x_1 u_1$ and hence $|x_0| = |x_1|$.

Let $z = z_0$ (we could also set $z = z_1$). We have $h(u) = h(z^m u)$ for every m . Note that $z \neq \varepsilon$. By replacing x_0 and x_1 by $z^m x_0$ and $z^m x_1$, respectively, for m large enough we can therefore assume that $|x_0| = |x_1| \geq |u|$. Let $z = z' z''$ with $|z'| + |u| = |x_0|$.

Consider an arbitrary bit string $\alpha = a_1 \cdots a_n \in \{0, 1\}^n$ of length n . We encode this bit string by the word $w(\alpha) = z_{a_1} z_{a_2} \cdots z_{a_n} z'$ of length $n' = \Theta(n)$. Let n' be the window size, let $\mathcal{A}_L^{n'}$ be the optimal (fixed-size) streaming algorithm from Section 4.1, and let r be the number of states of $\mathcal{A}_L^{n'}$, which implies $\text{space}_L^{\text{fs}}(n') = \lfloor \log_2 r \rfloor$. We claim that $\mathcal{A}_L^{n'}(w(\alpha)) \neq \mathcal{A}_L^{n'}(w(\beta))$ for $\alpha, \beta \in \{0, 1\}^n$ with $\alpha \neq \beta$. To see this, let $\alpha = a_1 \cdots a_n$ and $\beta = b_1 \cdots b_n$ of length n such that $\alpha \neq \beta$ but $\mathcal{A}_L^{n'}(w(\alpha)) = \mathcal{A}_L^{n'}(w(\beta))$. Let $1 \leq i \leq n$ be a position such that w.l.o.g. $a_i = 0$ and $b_i = 1$. Since $\mathcal{A}_L^{n'}(w(\alpha)) = \mathcal{A}_L^{n'}(w(\beta))$, we also have

$$\mathcal{A}_L^{n'}(w(\alpha) z'' z^{i-1} u) = \mathcal{A}_L^{n'}(w(\beta) z'' z^{i-1} u). \quad (6)$$

Note that $|z'' z^{i-1} u| = (i-1)|z| + |z''| + |u| = (i-1)|z| + |x_0| = (i-1)|z| + |x_1|$. The window contents after reading the words $w(\alpha) z'' z^{i-1} u$ and $w(\beta) z'' z^{i-1} u$ are:

$$\begin{aligned} u_0 z_{a_{i+1}} \cdots z_{a_n} z' z'' z^{i-1} u &= u_0 z_{a_{i+1}} \cdots z_{a_n} z^i u \\ u_1 z_{b_{i+1}} \cdots z_{b_n} z' z'' z^{i-1} u &= u_1 z_{b_{i+1}} \cdots z_{b_n} z^i u \end{aligned}$$

We have $h(u_0 z_{a_{i+1}} \cdots z_{a_n} z^i u) = h(u_0 u) \notin F$, i.e., $u_0 z_{a_{i+1}} \cdots z_{a_n} z^i u \notin L$ and $h(u_1 z_{b_{i+1}} \cdots z_{b_n} z^i u) = h(u_1 u) \in F$, i.e., $u_1 z_{b_{i+1}} \cdots z_{b_n} z^i u \in L$. Hence, we must have $w(\alpha) z'' z^{i-1} u \notin L(\mathcal{A}_L^{n'})$ and $w(\beta) z'' z^{i-1} u \in L(\mathcal{A}_L^{n'})$, which contradicts (6).

To sum up, we have shown that $\mathcal{A}_L^{n'}$ has at least 2^n many states, i.e., $r \geq 2^n$, which implies $\text{space}_L^{\text{fs}}(n') = \lfloor \log_2 r \rfloor \geq n$. Since n' and n are linearly related, we get $\text{space}_L^{\text{fs}}(n) \notin o(n)$. Moreover, $\text{space}_L^{\text{vs}}(n') \geq n$ together with the monotonicity of $\text{space}_L^{\text{vs}}(n)$ implies $\text{space}_L^{\text{vs}}(n) \in \Omega(n)$. \square

For a word $w \in \Sigma^*$ of length k we define the signature $\sigma(w) = b_1 \cdots b_k \in \{0, 1\}^*$ such that $b_i = 1$ if $h(w[i : k]) \in F$ and $b_i = 0$ otherwise. To complete our trichotomy, we finally show that languages in \mathcal{C}_2 are not streamable in space $o(\log n)$ in the fixed-size model.

Theorem 5.8. *If some SCC C of Γ is non-trivial and $\text{reach}_\Gamma(C)$ is not homomorphic to a directed cycle, then $\text{space}_L^{\text{fs}}(n) \notin o(\log n)$ and $\text{space}_L^{\text{vs}}(n) \in \Omega(\log n)$.*

Proof. Let C be the strongly connected component of Γ which is not trivial and where $\text{reach}_\Gamma(C)$ is not homomorphic to a directed cycle. Pick an arbitrary node $c \in C$. It must have indegree ≥ 1 since C is non-trivial. Since h is surjective there exists $u \in \Sigma^*$ with $h(u) = c$. We apply Lemma 5.6 to c and the subgraph $\text{reach}_\Gamma(C)$. This yields words $x, y \in \Sigma^+$ of equal length, say $k \geq 1$, which correspond to the paths π_0, π_1 in the lemma, such that $h(xu) \in F$ and $h(yu) \notin F$. Further, since C is strongly connected and non-trivial, there exists a non-trivial path π from c back to c , which yields a word $w \in \Sigma^+$ with $h(wu) = h(u)$. The situation is shown in Figure 3 on the right. Let $\ell = |w|$ and write k uniquely as $k = d \cdot \ell + (\ell - p + 1) = (d + 1) \cdot \ell - p + 1$ for $d \geq 0$ and $1 \leq p \leq \ell$. Consider the word $w[p : \ell] w^d$. In Γ this word yields the path consisting of d repetitions of the circle π followed by $\ell - p + 1$ more steps of π such that the whole path has length k . If $h(w[p : \ell] w^d u) \in F$ then we can replace x by $w[p : \ell] w^d$, otherwise we can replace y by $w[p : \ell] w^d$. Without loss of generality we can assume that $x = w[p : \ell] w^d$. From $h(xu) \in F$ and $h(yu) \notin F$ it follows that the signatures $\sigma(xu)$ and $\sigma(yu)$ differ in the first position. We can assume that for each position $i > 1$ we have $\sigma(xu)[i] = \sigma(yu)[i]$, otherwise we update the words x and y to the suffixes $x[i : k]$ and $y[i : k]$, respectively, where i is the maximal position such that $\sigma(xu)[i] \neq \sigma(yu)[i]$.

Consider now the n words $z_i = uw^{n-i} y w^i$ ($1 \leq i \leq n$) of equal length $n' = \ell \cdot n + |u| + |y| = \ell \cdot n + |u| + k \in \Theta(n)$. We now fix the window size to n' and consider the space optimal streaming algorithm $\mathcal{A}_L^{n'}$. We claim that for all i, j with $1 \leq i < j \leq n$ we have $\mathcal{A}_L^{n'}(z_i) \neq \mathcal{A}_L^{n'}(z_j)$. To deduce a contradiction, assume that $\mathcal{A}_L^{n'}(z_i) = \mathcal{A}_L^{n'}(z_j)$ with $1 \leq i < j \leq n$. This implies $\mathcal{A}_L^{n'}(z_i w^{n-i} u) = \mathcal{A}_L^{n'}(z_j w^{n-i} u)$. Moreover, after reading the input word $z_i w^{n-i} u = uw^{n-i} y w^n u$, the content of the sliding window

is $yw^n u$ (the suffix of $uw^{n-i}yw^n u$ of length $n' = \ell \cdot n + |u| + k$), which does not belong to L since $h(yw^n u) = h(yu) \notin F$. So, in order to get a contradiction, it suffices to show that the suffix of length n' of the input word $z_j w^{n-i} u = uw^{n-j} y w^{n+j-i} u$ belongs to L . We distinguish two cases (recall that $k = c \cdot \ell + (\ell - p + 1)$): If $j - i \geq c + 1$, then the suffix of $uw^{n-j} y w^{n+j-i} u$ of length n' is $w[p : \ell] w^{n+c} u = xw^n u$ which belongs to L since $h(xw^n u) = h(xu) \in F$. If $j - i \leq c$, then the suffix of $uw^{n-j} y w^{n+j-i} u$ of length $n' = \ell \cdot n + |u| + k$ is $y[1 + (j - i)\ell : k] w^{n+j-i} u$. We have $h(y[1 + (j - i)\ell : k] w^{n+j-i} u) = h(y[1 + (j - i)\ell : k] u)$. Now recall that the signatures $\sigma(xu)$ and $\sigma(yu)$ only differ in the first position. Since $1 + (j - i)\ell \geq 2$ it follows that $h(y[1 + (j - i)\ell : k] u) \in F$ if and only if $h(x[1 + (j - i)\ell : k] u) \in F$. Since $x = w[p : \ell] w^c$ and $j - i \leq c$ we have $x[1 + (j - i)\ell : k] = w[p : \ell] w^{c-j+i}$. Thus, we have $h(x[1 + (j - i)\ell : k] u) = h(w[p : \ell] w^{c-j+i} u) = h(w[p : \ell] w^c u) = h(xu) \in F$, which finally show that $y[1 + (j - i)\ell : k] w^{n+j-i} u$ belongs to L .

The above argument shows that $\mathcal{A}_L^{n'}$ has at least n states, and thus $\text{space}_L^{\text{fs}}(n') \geq \lfloor \log_2 n \rfloor$. Since n' and n are linearly related, we get $\text{space}_L^{\text{fs}}(n) \notin o(\log n)$. Moreover, $\text{space}_L^{\text{vs}}(n') \geq \lfloor \log_2 n \rfloor$ together with the monotonicity of $\text{space}_L^{\text{vs}}(n)$ implies $\text{space}_L^{\text{fs}}(n) \in \Omega(\log n)$. \square

5.3 Streaming regular languages and language growth

In this section, we present an alternative and quite natural characterization of the class of regular languages that are streamable in space $O(\log n)$. Note that by the results from Section 5.1 and Section 5.2, a regular language is streamable in space $O(\log n)$ in the fixed-size model if and only if it is streamable in space $O(\log n)$ in the variable-size model. Our characterization is based on (i) language growth and (ii) reductions that are computable by Mealy machines.

A *Mealy machine* $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$ consists of a finite set of states Q , an input alphabet Σ , an output alphabet Γ , an initial state $q_0 \in Q$ and the transition function $\delta : Q \times \Sigma \rightarrow Q \times \Gamma$. We extend the transition function to $\delta : Q \times \Sigma^* \rightarrow Q \times \Gamma^*$ in the usual way: $\delta(q, \varepsilon) = (q, \varepsilon)$ and $\delta(q, ua) = (p, vb)$, where $\delta(q, u) = (r, v)$ and $\delta(r, a) = (p, b)$. The function $f_{\mathcal{M}} : \Sigma^* \rightarrow \Gamma^*$ computed by \mathcal{M} is defined by $\delta(q_0, u) = (q, f_{\mathcal{M}}(u))$ for some (unique) state q . Note that $f_{\mathcal{M}}$ is length-preserving. A function $f : \Sigma^* \rightarrow \Gamma^*$ is called a \leftarrow -transduction if there exists a Mealy machine \mathcal{M} such that $f(u) = \text{rev}(f_{\mathcal{M}}(\text{rev}(u)))$. For instance, the function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ which inverts the last bit is a \leftarrow -transduction, but inverting the first bit is not. Notice that \leftarrow -transductions are *suffix-preserving*, i.e. if $f(v') = w'$, then $f(vv') = ww'$ for some w . For languages $K \subseteq \Sigma^*$, $L \subseteq \Gamma^*$, we say that K is \leftarrow -reducible to L , if there exists a \leftarrow -transduction $f : \Sigma^* \rightarrow \Gamma^*$ such that $x \in K$ if and only if $f(x) \in L$ for all $x \in \Sigma^*$.

Lemma 5.9. *If K is \leftarrow -reducible to L via a Mealy machine with d states, then $\text{space}_K^{\text{vs}}(n) \leq d \cdot (\text{space}_L^{\text{vs}}(n) + 1)$.*

Proof. Assume that K is \leftarrow -reducible to L via the \leftarrow -transduction f computed by a Mealy machine $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$. Further, for every state $q \in Q$ let f_q be the \leftarrow -transduction computed by the Mealy machine $\mathcal{M}_q = (Q, \Sigma, \Gamma, q, \delta)$.

Recall the definition of $\text{space}_L^{\text{vs}}(n)$. Our goal is to show that $|\psi_K(\Sigma^{\leq n})| \leq |\psi_L(\Gamma^{\leq n})|^{|Q|}$ for all n , which then implies

$$\lfloor \log_2 |\psi_K(\Sigma^{\leq n})| \rfloor \leq \lfloor |Q| \cdot \log_2 |\psi_L(\Sigma^{\leq n})| \rfloor \leq |Q| \cdot (\lfloor \log_2 |\psi_L(\Sigma^{\leq n})| \rfloor + 1).$$

Given a word $w \in \Sigma^*$ we define $F(w)$ to be the function $Q \rightarrow (\Gamma^*/\sim_L)^*$ which maps a state $q \in Q$ to $\psi_L(f_q(w))$. We show that the function

$$\begin{aligned} T : F(\Sigma^{\leq n}) &\rightarrow \psi_K(\Sigma^{\leq n}) \\ F(x) &\mapsto \psi_K(x) \end{aligned}$$

is well-defined and (clearly) surjective, which implies that $|\psi_K(\Sigma^{\leq n})| \leq |F(\Sigma^{\leq n})| \leq |\psi_L(\Gamma^{\leq n})|^{|Q|}$.

Assume that $a_1 \cdots a_k, b_1 \cdots b_k \in \Sigma^{\leq n}$ satisfy $F(a_1 \cdots a_k) = F(b_1 \cdots b_k)$, i.e., $\psi_L(f_q(a_1 \cdots a_k)) = \psi_L(f_q(b_1 \cdots b_k))$ for all $q \in Q$. We need to show that $\psi_K(a_1 \cdots a_k) = \psi_K(b_1 \cdots b_k)$, i.e., for all $1 \leq i \leq k$ and all $x \in \Sigma^*$ we have

$$a_i \cdots a_k x \in K \iff b_i \cdots b_k x \in K.$$

Define $(q, y) = \delta(q_0, \text{rev}(x))$ and let $z = \text{rev}(y)$. Since $F(a_1 \cdots a_k) = F(b_1 \cdots b_k)$, we have

$$\psi_L(f_q(a_1 \cdots a_k)) = \psi_L(f_q(b_1 \cdots b_k)).$$

and in particular $f_q(a_i \cdots a_k) \sim_L f_q(b_i \cdots b_k)$ because f_q is suffix-preserving. Summarizing, we have

$$\begin{aligned} a_i \cdots a_k x \in K &\iff f(a_i \cdots a_k x) \in L \\ &\iff f_q(a_i \cdots a_k)z \in L \\ &\iff f_q(b_i \cdots b_k)z \in L \\ &\iff f(b_i \cdots b_k x) \in L \\ &\iff b_i \cdots b_k x \in K. \end{aligned}$$

Finally, T is clearly surjective, which concludes the proof. \square

Lemma 5.10. *If $L \subseteq \Sigma^*$ is regular, then ψ_L is a \leftarrow -transduction. Hence, L is \leftarrow -reducible to $\psi_L(L)$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ the minimal DFA for L and M be the transition monoid of \mathcal{A} . It is the submonoid of Q^Q (the set of all mappings on Q under composition) generated by all mappings f_a ($a \in \Sigma$) with $f_a(q) = \delta(q, a)$. By the Myhill-Nerode Theorem we can identify the

equivalence classes in Σ^*/\sim_L with the states in Q . Hence, we can regard $\psi_L(\Sigma)^*$ as a language over the finite alphabet Q . Define the Mealy machine $\mathcal{M} = (M, \Sigma, Q, 1, \mu)$ where $\mu(m, a) = (f_a \circ m, q_0(f_a \circ m))$ for all $m \in M$, $a \in \Sigma$ (here, $f_a \circ m$ is the mapping obtained by first applying f_a followed by m , and $q_0(f_a \circ m) = m(f_a(q))$). We clearly have $\psi_L(w) = \text{rev}(f_{\mathcal{M}}(\text{rev}(w)))$. \square

Lemma 5.10 also implies that $\psi_L(L)$ is regular if L is regular.

Theorem 5.11. *Let $K \subseteq \Sigma^*$ be a regular language. Then $\text{space}_K^{\text{vs}}(n) \in O(\log n)$ if and only if K is \leftarrow -reducible to a regular language L of polynomial growth.*

Proof. First, $\text{space}_K^{\text{vs}}(n) \in O(\log n)$ implies by its definition that $\psi_K(\Sigma^*)$ has polynomial growth. Hence, also $\psi_K(K) \subseteq \psi_K(\Sigma^*)$ has polynomial growth. By Lemma 5.10, K is \leftarrow -reducible to $\psi_K(K)$ and the latter is regular.

Conversely, if L is regular and has polynomial growth, then $\text{space}_L^{\text{fs}}(n) \in O(\log n)$ by Theorem 4.3. By the results from the previous section, this implies $\text{space}_L^{\text{vs}}(n) \in O(\log n)$. By Lemma 5.9 we conclude $\text{space}_K^{\text{vs}}(n) \in O(\log n)$. \square

6 Randomized streaming

In this section, we consider randomized streaming algorithms. We will restrict to the fixed-size model. We will show that any space lower bound for the fixed-size model also holds in the randomized Monte-Carlo setting.

A *randomized streaming algorithm* (for the fixed-size model) is a pair $\mathcal{A} = ((\mathcal{A}_\rho)_{\rho \in \mathbb{N}}, (\mathcal{P}_m)_{m \in \mathbb{N}})$ such that:

- every \mathcal{A}_ρ is a streaming algorithm $\mathcal{A}_\rho = (S_\rho, \Sigma, \Phi_\rho, s_{\rho,0}, F_\rho)$ for the fixed-size model as defined in Section 3.1, and
- for every $m \geq 0$, \mathcal{P}_m is a probability distribution on a finite non-empty subset R_m of \mathbb{N} , i.e., $\sum_{\rho \in R_m} \mathcal{P}_m(\rho) = 1$ and $0 \leq \mathcal{P}_m(\rho) \leq 1$ for all $\rho \in R_m$.

The idea is that R_m is the set of random numbers used for input streams of length m . W.l.o.g. we can assume that the sets R_m partition \mathbb{N} . Thus every number $n \in \mathbb{N}$ belongs to a unique R_m . Then, given an input word $w \in \Sigma^*$ of length m , the algorithm first randomly, according to the distribution \mathcal{P}_m , chooses a random number $\rho \in R_m$ and runs the streaming algorithm \mathcal{A}_ρ on w .

We want to define the average space consumption and error probability of a randomized streaming algorithm on an input string $w \in \Sigma^*$. Let us first

assume that \mathcal{B} is a (non-randomized) streaming algorithm and let $w \in \Sigma^m$ be an input of length m . We define

$$\text{space}^{\text{fs}}(\mathcal{B}, w) = \max\{|\mathcal{B}(u)| : u \in \text{prefix}(w)\} \quad (7)$$

as the maximal space consumption of \mathcal{B} on w . Moreover, for a language $L \subseteq \Sigma^*$ and a window size $n \in \mathbb{N}$ let

$$\chi_L^n(\mathcal{B}, w) = \begin{cases} 0 & \text{if } (v \in L(\mathcal{B}) \iff \text{last}_n(v) \in L) \text{ for all } v \in \text{prefix}(w) \\ 1 & \text{else.} \end{cases}$$

Thus, $\chi_L^n(\mathcal{B}, w)$ is zero, if for random number ρ and input stream w , the algorithm \mathcal{B} correctly decides at every time instant, whether the current window content (assuming window size n) belongs to L .

We now define the average space consumption of the randomized algorithm on w as

$$\text{space}^{\text{fs}}(\mathcal{A}, w) = \sum_{\rho \in R_m} \mathcal{P}_m(\rho) \cdot \text{space}^{\text{fs}}(\mathcal{A}_\rho, w).$$

The error probability of the randomized algorithm for the input stream $w \in \Sigma^*$ and window size n is

$$\chi_L^n(\mathcal{A}, w) = \sum_{\rho \in R_m} \mathcal{P}_m(\rho) \cdot \chi_L^n(\mathcal{A}_\rho, w).$$

For a real number $\tau \in [0, 1]$, the randomized algorithm \mathcal{A} is τ -correct (for the language L and window size n) if $\chi_L^n(\mathcal{A}, w) \leq \tau$ for all inputs $w \in \Sigma^*$ (of arbitrary length). We say that L is *Las-Vegas-streamable* (resp., *Monte-Carlo-streamable*) in space $s(n)$ in the fixed-size model for every window size n there exists a 0-correct (resp. 1/3-correct) randomized algorithm as described above such that $\text{space}^{\text{fs}}(\mathcal{A}, w) \leq s(n)$ for every input stream $w \in \Sigma^*$. The bound 1/3 for the Monte-Carlo setting is arbitrary; any constant $c < 1/2$ would work as well.

Using Yao's min-max principle [29], one can use the argument from the proof of Theorem 5.7 to show that if some SCC of Γ is not homomorphic to a directed cycle, then L is not Las-Vegas-streamable in space $o(n)$ in the fixed-size model. Here, we will prove the same lower bound for the more general Monte-Carlo setting. In fact, we show that up to a constant factor, any lower bound for deterministic streaming algorithms transfers to the Monte-Carlo setting. Recall the definition of $\text{space}_L^{\text{fs}}(n)$ from (3). It is the space consumption of an optimal streaming algorithm for L and window size n .

Theorem 6.1. *Let $0 \leq \tau < 1$ and let \mathcal{A} be a τ -correct randomized algorithm for the language $L \subseteq \Sigma^*$ and window size n . Then there exists an input string $w \in \Sigma^*$ such that $\text{space}^{\text{fs}}(\mathcal{A}, w) \geq (1 - \tau) \cdot \text{space}_L^{\text{fs}}(n)$.*

For the proof of Theorem 6.1, we need a lemma that allows to bound the size of the storage content of a randomized streaming algorithm. For deterministic streaming algorithm \mathcal{A} , one can clearly assume that all storage contents of the streaming algorithm for window size n have length at most n : If some storage content of length larger than n is reachable from s_0 , then one can replace \mathcal{A} by the trivial DFA \mathcal{B}_L^n that uses at most space n . This argument does not work for in the Monte-Carlo setting. Despite the existence of an input stream w and random number ρ with $\text{space}^{\text{fs}}(\mathcal{A}_\rho, w) > n$, the average space consumption $\text{space}^{\text{fs}}(\mathcal{A}, w)$ might be in $o(n)$. On the other hand, for the proof of Theorem 6.1 we only need some bound on the length of the storage contents that only depends on the window size n .

Lemma 6.2. *Let $L \subseteq \Sigma^*$. There exists a mapping $f(n) \in 2^{O(n)}$ such that the following holds for every window size n : For every (non-randomized) streaming algorithm \mathcal{A} there exists a streaming algorithm \mathcal{A}' such that the following holds for all inputs $w \in \Sigma^*$:*

- (i) *the set of states of \mathcal{A}' is a subset of $\{0, 1\}^{\leq f(n)}$,*
- (ii) $\text{space}^{\text{fs}}(\mathcal{A}', w) \leq \text{space}^{\text{fs}}(\mathcal{A}, w)$,
- (iii) $\chi_L^n(\mathcal{A}', w) \leq \chi_L^n(\mathcal{A}, w)$ (which means that \mathcal{A}' works correctly, whenever \mathcal{A} works correctly).

Proof. Assume that $|\Sigma| = c$. The mapping $f(n)$ is defined by

$$f(n) = \left\lceil \log_2 \left(c^n + \frac{(c^n - 1) \cdot 2^{(c^n - 1)/(c - 1)}}{c - 1} \right) \right\rceil. \quad (8)$$

Let $\mathcal{A} = (S, \Sigma, \Phi, s_0, F)$. We partition the state set S as $S = Q_{\leq} \uplus Q_{>}$, where $Q_{\leq} = \{s \in S : |s| \leq f(n)\}$ and $Q_{>} = \{s \in S : |s| > f(n)\}$. W.l.o.g. we can assume that the initial state s_0 belongs to Q_{\leq} .

Next, we assign to each state $s \in Q_{>}$ the set

$$\lambda(s) := \{w \in \Sigma^* : |w| \leq n - 1, \Phi(s, w) \in F\} \subseteq \Sigma^{\leq n-1}.$$

Thus, $\lambda(S)$ contains all input words of length at most $n - 1$ that lead from a to a final state. Now, we define the DFA

$$\mathcal{A}' := (S', \Sigma, \Phi', s_0, F'),$$

as follows:

- The state set is $S' = Q_{\leq} \uplus (2^{\Sigma^{\leq n-1}} \times \Sigma^{\leq n-1}) \uplus \Sigma^n$.
- The transition function Φ' of \mathcal{A}' is defined as follows: For $s \in Q_{\leq}$ we set

$$\Phi'(s, a) = \begin{cases} \Phi(s, a) & \text{if } \Phi(s, a) \in Q_{\leq}, \\ \lambda(\Phi(s, a)) & \text{else.} \end{cases}$$

Now assume that $K \subseteq \Sigma^{\leq n-1}$ and $u \in \Sigma^{\leq n-1}$. Then we set

$$\Phi'((K, u), a) = \begin{cases} (K, ua) & \text{if } |u| < n-1, \\ (ua, \varepsilon) & \text{else.} \end{cases}$$

Finally, for $u = bv$ with $b \in \Sigma$, $v \in \Sigma^{n-1}$ we set $\Phi'(u, a) = va$.

- The initial state is still $s_0 \in Q_{\leq}$.
- The set of final states is

$$F' = (Q_{\leq} \cap F) \uplus \{(K, u) : K \subseteq \Sigma^{\leq n-1}, u \in K\} \uplus (\Sigma^n \cap L).$$

Note that

$$|(2^{\Sigma^{\leq n-1}} \times \Sigma^{\leq n-1}) \uplus \Sigma^n| = c^n + \frac{(c^n - 1) \cdot 2^{(c^n - 1)/(c-1)}}{c - 1}.$$

The definition of $f(n)$ from (8) implies that every state from $(2^{\Sigma^{\leq n-1}} \times \Sigma^{\leq n-1}) \uplus \Sigma^n$ can be encoded by a bit string of length exactly $f(n)$. After this, the set of states of \mathcal{A}' consists of bit strings of length at most $f(n)$, which yields property (i).

For (ii) note that $\text{space}^{\text{fs}}(\mathcal{A}, w)$ is the maximal length of a storage content (or state), when reading the input word w in the automaton \mathcal{A} starting from s_0 . Let s_0, s_1, \dots, s_m (where $m = |w|$) the sequence of visited states. If all s_i belong to Q_{\leq} , then the same sequence of states is visited when reading the input word w in the automaton \mathcal{A}' starting from s_0 . Hence, in this case we have $\text{space}^{\text{fs}}(\mathcal{A}, w) = \text{space}^{\text{fs}}(\mathcal{A}', w)$. On the other hand, if some s_i belongs to $Q_{>}$, then $\text{space}^{\text{fs}}(\mathcal{A}, w) > f(n)$, while $\text{space}^{\text{fs}}(\mathcal{A}', w) \leq f(n)$ is ensured by the construction of \mathcal{A}' .

For property (iii) we can argue similarly. Let the sequence s_0, s_1, \dots, s_m be as above and let $s'_0 = s_0, s'_1, \dots, s'_m$ be the sequence of states visited in \mathcal{A}' . If all s_i belong to Q_{\leq} then $s_i = s'_i$ for all $0 \leq i \leq m$ and we get $\chi_L^n(\mathcal{A}, w) = \chi_L^n(\mathcal{A}', w)$. On the other hand, if s_i is the first storage content in $Q_{>}$, then the construction of \mathcal{A}' implies that $s_j \in F$ if and only if $s'_j \in F'$ for all $0 \leq j \leq i + n - 1$: For $0 \leq j \leq i - 1$ this is obvious, since $s_j = s'_j$. For $i \leq j \leq i + n - 1$, the new states from $2^{\Sigma^{\leq n-1}} \times \Sigma^{\leq n-1}$ ensure that $s_j \in F$ if and only if $s'_j \in F'$. For $i + n \leq j \leq m$, this equivalence is no longer guaranteed, but note that the states s'_{i+n}, \dots, s'_m explicitly store the current window content (a string from Σ^n). Hence, \mathcal{A}' gives a correct answer for all time instants j with $i + n \leq j \leq m$. \square

By applying for a given randomized streaming algorithm \mathcal{A} Lemma 6.2 to every \mathcal{A}_ρ we obtain:

Lemma 6.3. *Let $L \subseteq \Sigma^*$. There exists a mapping $f(n) \in 2^{O(n)}$ such that the following holds for every window size n : For every randomized streaming algorithm \mathcal{A} there exists a randomized streaming algorithm \mathcal{A}' such that the following holds for all inputs $w \in \Sigma^*$:*

- (i) *for every $\rho \in \mathbb{N}$, the set of states of \mathcal{A}'_ρ is a subset of $\{0, 1\}^{\leq f(n)}$,*
- (ii) $\text{space}^{\text{fs}}(\mathcal{A}', w) \leq \text{space}^{\text{fs}}(\mathcal{A}, w)$,
- (iii) $\chi_L^n(\mathcal{A}', w) \leq \chi_L^n(\mathcal{A}, w)$.

Note that the randomized algorithm \mathcal{A}' from Lemma 6.2 is at least as good as \mathcal{A} with respect to space and error probability.

Proof of Theorem 6.1. Fix a window size n and let \mathcal{A}_L^n be the minimal DFA defined at the end of Section 3.1. It is an optimal streaming algorithm for L and window size n . Let us write this DFA as $\mathcal{A}_L^n = (P, \Sigma, \delta, p_0, F)$ and let $P = \{p_0, p_1, \dots, p_{r-1}\}$. Thus $\text{space}_L^{\text{fs}}(n) = \lfloor \log_2 r \rfloor$. Note that the directed graph underlying \mathcal{A}_L^n is strongly connected, i.e., from every state one can reach every other state (this property is true for the automaton \mathcal{B}_L^n and is preserved by minimization). Since \mathcal{A}_L^n is a minimal DFA, for all $0 \leq i < j < r$ there exists a word $y_{i,j}$ such that $\delta(p_i, y_{i,j}) \in F$ if and only if $\delta(p_j, y_{i,j}) \notin F$.

Let $\mathcal{A} = ((\mathcal{A}_\rho)_{\rho \in \mathbb{N}}, (\mathcal{P}_m)_{m \in \mathbb{N}})$ be a τ -correct randomized streaming algorithm for L and window size n . We have to show that there exists an input string $w \in \Sigma^*$ such that $\text{space}^{\text{fs}}(\mathcal{A}, w) \geq (1 - \tau) \cdot \text{space}_L^{\text{fs}}(n)$. Let $Q = \{0, 1\}^{\leq f(n)}$. By Lemma 6.3 we can assume that the state set S_ρ of \mathcal{A}_ρ is a subset of Q . By adding dummy states, we can assume that $S_\rho = Q$. Let $\mathcal{B}_1, \dots, \mathcal{B}_k$ be an enumeration of all DFA with state set Q and input alphabet Σ (or equivalently, streaming algorithms with set of storage contents Q and inputs alphabet Σ). Every \mathcal{A}_ρ (when viewed as a DFA) appears in this list. Consider the product DFA

$$\widehat{\mathcal{A}} = \mathcal{A}_L^n \times \prod_{i=1}^k \mathcal{B}_i,$$

and let us write $\widehat{\mathcal{A}} = (P \times Q^k, \Sigma, \widehat{\delta}, \widehat{q}_0, \widehat{F})$. Let $C \subseteq P \times Q^k$ be a maximal SCC in the directed graph underlying $\widehat{\mathcal{A}}$ such that moreover C is reachable from the initial state \widehat{q}_0 . We choose an arbitrary state $\widehat{q} = (p, \dots) \in C$. Thus, $p \in P$ is the \mathcal{A}_L^n -component of \widehat{q} . Let $v_0 \in \Sigma$ be word such that $\widehat{\delta}(\widehat{q}_0, v_0) = \widehat{q}$. Moreover, for $0 \leq i < r$ let $x_i \in \Sigma^*$ such that $\delta(p, x_i) = p_i$ (recall that \mathcal{A}_L^n is strongly connected). We now define a word $w \in \Sigma^*$ as

$$w = v_0 \prod_{0 \leq i < j < r} x_i y_{i,j} v_{i,j} x_j y_{i,j} v'_{i,j} \quad (9)$$

Here the words $v_{i,j}$ and $v'_{i,j}$ are chosen such that if w' is a prefix of w that ends with one of the displayed copies of $v_{i,j}$ and $v'_{i,j}$ in (9), then $\widehat{\delta}(\widehat{q}_0, w') = \widehat{q}$. These words exist, since the prefix v_0 of w leads to the maximal SCC C and $\widehat{q} \in C$. Hence, for every $0 \leq i < r$ there exists a state $\widehat{q}_i \in C$ such that $\widehat{\delta}(\widehat{q}_0, w') = \widehat{q}_i$ for every prefix of w that ends with one of the $(r-1)$ many displayed occurrences of x_i in (9).

We claim that for every $1 \leq l \leq k$, if $\chi_L^n(\mathcal{B}_l, w) = 0$ (i.e., \mathcal{B}_l is a correct streaming algorithm for L and window size n on input w), then \mathcal{B}_l visits at least r many different states while reading w . More precisely, we claim that if $\widehat{q}_{i,l}$ is the \mathcal{B}_l -component of the $\widehat{\mathcal{A}}$ -state \widehat{q}_i , then $\widehat{q}_{i,l} \neq \widehat{q}_{j,l}$ for $0 \leq i, j < r$, $i \neq j$. To see this, assume that there exist $0 \leq i < j < r$ such that $\widehat{q}_{i,l} = \widehat{q}_{j,l}$. Let δ_l be the transition mapping of \mathcal{B}_l and $q_{0,l}$ its initial state. Now consider the two prefixes w', w'' of w that end with the two displayed occurrences of $y_{i,j}$ in (9). Then, $\delta_l(q_{0,l}, w') = \delta_l(\widehat{q}_{i,l}, y_{i,j}) = \delta_l(\widehat{q}_{j,l}, y_{i,j}) = \delta_l(q_{0,l}, w'')$. On the other hand, in the DFA \mathcal{A}_L^n we have $\delta(p_0, w') = \delta(p_i, y_{i,j})$ and $\delta(p_0, w'') = \delta(p_j, y_{i,j})$. Moreover, $\delta(p_i, y_{i,j}) \in F$ if and only if $\delta(p_j, y_{i,j}) \notin F$. Assume w.l.o.g. that $\delta(p_0, w') \in F$ if and only if $\delta_l(q_{0,l}, w') \notin F$. But this contradicts the fact that \mathcal{B}_l and \mathcal{A}_L^n are both correct on input w .

Let $m = |w|$. Since \mathcal{A} is τ -correct we have

$$\chi_L^n(\mathcal{A}, w) = \sum_{\rho \in R_m} \mathcal{P}_m(\rho) \cdot \chi_L^n(\mathcal{A}_\rho, w) \leq \tau$$

Let $U_m = \{\rho \in R_m : \chi_L^n(\mathcal{A}_\rho, w) = 0\}$, which is the set of all random numbers $\rho \in R_m$ such that \mathcal{A} with random number ρ is correct on input w . We get

$$\sum_{\rho \in U_m} \mathcal{P}_m(\rho) \geq 1 - \tau.$$

Moreover, since for every $\rho \in U_m$, \mathcal{A}_ρ is correct on w , the above claim implies that \mathcal{A}_ρ visits at least r many different storage contents while reading w . This implies $\text{space}^{\text{fs}}(\mathcal{A}_\rho, w) \geq \lceil \log_2 r \rceil = \text{space}_L^{\text{fs}}(n)$. We get

$$\begin{aligned} \text{space}^{\text{fs}}(\mathcal{A}, w) &= \sum_{\rho \in R_m} \mathcal{P}_m(\rho) \cdot \text{space}^{\text{fs}}(\mathcal{A}_\rho, w) \\ &\geq \sum_{\rho \in U_m} \mathcal{P}_m(\rho) \cdot \text{space}^{\text{fs}}(\mathcal{A}_\rho, w) \\ &\geq \sum_{\rho \in U_m} \mathcal{P}_m(\rho) \cdot \text{space}_L^{\text{fs}}(n) \\ &\geq (1 - \tau) \cdot \text{space}_L^{\text{fs}}(n). \end{aligned}$$

This proves Theorem 6.1. \square

It is remarkable that our proof of Theorem 6.1 does not use Yao's min-max principle.

From Theorem 6.1 and the lower bounds in Theorem 5.7 and 5.8 we obtain:

Corollary 6.4. *The following hold for every regular language L with associated colored left Cayley graph Γ :*

- *If some SCC C of Γ is not homomorphic to a directed cycle, then L is not Monte-Carlo-streamable in space $o(n)$.*
- *If some SCC C of Γ is non-trivial and $\text{reach}_\Gamma(C)$ is not homomorphic to a directed cycle, then L is not Monte-Carlo-streamable in space $o(\log n)$.*

7 Computing streaming space complexity

In this section we show that in nondeterministic logarithmic space (and hence in polynomial time) one can check whether a regular language that is given by a DFA is streamable in space $O(1)$ in the fixed-size model. Furthermore we prove the same complexity to test whether a given regular language is streamable in $O(\log n)$ in the variable-size model (and hence also in the fixed-size model). By minimizing the input DFA, we can assume that the input DFA \mathcal{A} is the minimal DFA for $L = L(\mathcal{A}) \subseteq \Sigma^*$.

Theorem 7.1. *Given a minimal DFA \mathcal{A} , one can test in nondeterministic logspace whether $L(\mathcal{A})$ is streamable in space $O(1)$ in the fixed-size model.*

Proof. By Corollary 4.8, $L(\mathcal{A})$ is not streamable in space $O(1)$ in the fixed-size model if and only if there exist words $s, s', t \in \Sigma^*$ such that $|s| = |s'|$, $|t| = |Q|$ and $\mathcal{A}(st) \neq \mathcal{A}(s't)$. The existence of such words can be easily verified in nondeterministic logspace: One simulates \mathcal{A} on two words of the same length (the words s, s'), and thereby only stores the current state pair. At every time instant, the algorithm can nondeterministically decide to continue the simulation from the current state pair (p, q) with a single word (the word t) for $|Q|$ steps. The algorithm accepts if at the end the two states are different.

Since nondeterministic logspace is closed under complement, one can also decide in nondeterministic logspace whether $L(\mathcal{A})$ is streamable in space $O(1)$ in the fixed-size model. \square

Another interesting corollary of Lemma 4.5 is that if $L(\mathcal{A})$ is streamable in space $O(1)$, then there is also a space efficient streaming algorithm in the uniform setting:

Theorem 7.2. *The algorithm in Figure 4 solves the streaming problem in the fixed-size model for a given m -state DFA \mathcal{A} and a given window size n (which are both part of the input), assuming that $L(\mathcal{A})$ is streamable in constant space. Apart from the input DFA, the algorithm uses space $m \cdot \lceil \log_2 |\Sigma| \rceil + \lceil \log_2 m \rceil$.*

INITIALIZE($\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, $n \in \mathbb{N}$)	
$q \leftarrow \mathcal{A}(\square^{\max\{n- Q , 0\}})$	\triangleright the initial window is \square^n
$w \leftarrow \square^{\min\{n, Q \}}$	\triangleright maintain suffix of length $\min\{n, Q \}$
UPDATE($a \in \Sigma$)	
$w \leftarrow w[2:]a$	
TEST	
return true if $\delta(q, w) \in F$	

Figure 4: The constant space streaming algorithm. INITIALIZE receives as an input a DFA \mathcal{A} and a window length n . UPDATE is called each time a symbol is inserted into the stream. The window belongs to the accepted language iff TEST returns true.

Proof. Recall from Section 3.1 that the window is initially filled with \square^n where $\square \in \Sigma$ is an arbitrary symbol. By Lemma 4.5 it suffices to store the maximal suffix of length at most $|Q|$, which is stored in w . The function UPDATE removes the first symbol of w and adds a new symbol. To test whether the current window content belongs to the language, the automaton is simulated on the word vw of length n where $v \in \square^*$. \square

In the rest of the section, we show that one can also decide in non-deterministic logspace whether $L(\mathcal{A})$ is streamable in space $O(\log n)$ in the variable-size model (and hence in the fixed-size model). For words $u, x_0, x_1 \in \Sigma^*$ we define

$$Q(u, x_0, x_1) = \{\mathcal{A}(ux) : x \in \{x_0, x_1\}^*\},$$

which is the set of states of \mathcal{A} reachable from the initial state by first reading u and then an arbitrary product of copies of x_0 and x_1 .

Lemma 7.3. *The language L is not streamable in space $o(n)$ in the variable-size model if and only if there are words $u_0, u_1, v_0, v_1 \in \Sigma^+$ such that $|u_0| = |u_1|$ and $Q(u_0, v_0u_0, v_1u_1) \cap Q(u_1, v_0u_0, v_1u_1) = \emptyset$.*

Proof. Let us first assume that L is not streamable in space $o(n)$ in the variable-size model. Recall the proof of Theorem 5.7. There are words $u_0, u_1, v_0, v_1 \in \Sigma^+$ and $u \in \Sigma^*$ such that

- $|u_0| = |u_1|$,
- $h(v_0u_0u) = h(u) = h(v_1u_1u)$,
- $h(u_0u) \notin F$ and $h(u_1u) \in F$.

Setting $K = \{v_0u_0, v_1u_1\}^*$ we have $u_0Ku \cap L = \emptyset$ and $u_1Ku \subseteq L$. Hence for all $w_0 \in u_0K$ and $w_1 \in u_1K$ we have $w_0 \not\sim_L w_1$. Since \mathcal{A} is minimal, this

implies $\{\mathcal{A}(w) : w \in u_0K\} \cap \{\mathcal{A}(w) : w \in u_1K\} = \emptyset$, i.e., $Q(u_0, v_0u_0, v_1u_1) \cap Q(u_1, v_0u_0, v_1u_1) = \emptyset$.

Next, assume that $Q(u_0, v_0u_0, v_1u_1) \cap Q(u_1, v_0u_0, v_1u_1) = \emptyset$ and $|u_0| = |u_1|$. We clearly have $u_0 \neq u_1$ and hence $v_0u_0 \neq v_1u_1$. Further, we can choose numbers $p, q \geq 1$ such that $(v_0u_0)^p$ and $(v_1u_1)^q$ have the same length. We redefine v_0 to be $(v_0u_0)^{p-1}v_0$ and v_1 to be $(v_1u_1)^{q-1}v_1$ and one can verify that the new resulting sets $Q(u_i, v_0u_0, v_1u_1)$ are contained in the original sets, and are therefore also disjoint.

Now consider a streaming algorithm for L (with respect to the variable-size model) and let n be arbitrary. We claim that for all $w_1, w_2 \in \{v_0u_0, v_1u_1\}^n$ with $w_1 \neq w_2$, pushing w_1 and w_2 leads to different storage contents. This is because after popping a suitable number of symbols, the two windows contain words $x_0 \in u_0\{v_0u_0, v_1u_1\}^*$ and $x_1 \in u_1\{v_0u_0, v_1u_1\}^*$. By assumption, reading x_0 and x_1 in the minimal DFA \mathcal{A} leads to different states. Hence there exists a word $z \in \Sigma^*$ such that $xz \in L$ if and only if $yz \notin L$. This proves that the algorithm cannot work in space $o(n)$. \square

We call a tuple (u_0, u_1, w_0, w_1) of words *critical*, if $|u_0| = |u_1|$, u_i is a suffix of w_i for all $i \in \{0, 1\}$ and $Q(u_0, w_0, w_1) \cap Q(u_1, w_0, w_1) = \emptyset$. Clearly, the condition from Lemma 7.3 is equivalent to the existence of a critical tuple.

Lemma 7.4. *If there exists a critical tuple, then there exists a critical tuple (u_0, u_1, w_0, w_1) such that $Q(u_0, w_0, w_1)$ and $Q(u_1, w_0, w_1)$ have each size at most three.*

Proof. Let $h : \Sigma^* \rightarrow M$ be the canonical morphism into the transition monoid M of \mathcal{A} , which acts on Q via $M \times Q \rightarrow Q$, $m \mapsto q \cdot m = m(q)$. Assume that (u_0, u_1, w_0, w_1) is a critical tuple. Notice that

$$Q(u_i, w_0, w_1) = \{\delta(q_0, u_i) \cdot m : m \in \{h(w_0), h(w_1)\}^*\}$$

where X^* denotes the submonoid generated by a set $X \subseteq M$. It suffices to define a new critical tuple (u_0, u_1, x_0, x_1) with the property that $h(x_i) \cdot h(x_j) = h(x_j)$ for all $i, j \in \{0, 1\}$. This implies $\{h(x_0), h(x_1)\}^* = \{1, h(x_0), h(x_1)\}$, and hence, $Q(u_i, x_0, x_1)$ contains at most three elements for both $i \in \{0, 1\}$.

Notice that if (u_0, u_1, w_0, w_1) is critical, then also $(u_0, u_1, y_0w_0, y_1w_1)$ is critical for all $y_0, y_1 \in \{w_0, w_1\}^*$. Let $\omega \geq 1$ be a number such that m^ω is idempotent for all $m \in M$. By choosing $e_0 = (h(w_0)^\omega h(w_1)^\omega)^\omega h(w_0)^\omega$ and $e_1 = (h(w_0)^\omega h(w_1)^\omega)^\omega$ we indeed obtain $e_i e_j = e_j$ for all $i, j \in \{0, 1\}$. Hence we define $x_0 = (w_0^\omega w_1^\omega)^\omega w_0^\omega$ and $x_1 = (w_0^\omega w_1^\omega)^\omega$. \square

Lemma 7.5. *Given a minimal DFA \mathcal{A} , one can test in nondeterministic logspace whether \mathcal{A} has a critical tuple.*

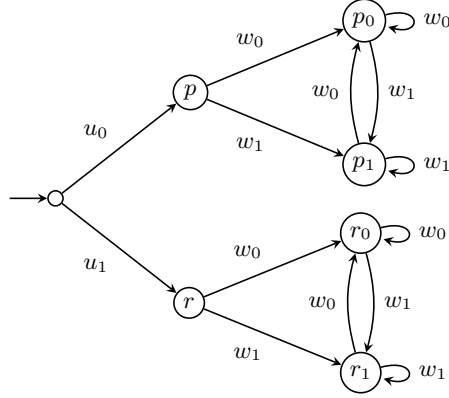


Figure 5: A critical tuple (u_0, u_1, w_0, w_1) .

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a minimal DFA. Figure 5 illustrates the structure we need to detect in \mathcal{A} . To do so, we reduce to testing emptiness of one-counter automata, which is known to be decidable in nondeterministic logspace [24]. For two states $p, r \in Q$ let $\mathcal{A}_{p,r} = (Q, \Sigma, \delta, p, \{r\})$, i.e., the automaton \mathcal{A} with initial state p and final state r , and let $L(p, r) = L(\mathcal{A}_{p,r})$.

The algorithm iterates over all disjoint sets $\{p, p_0, p_1\}, \{r, r_0, r_1\} \subseteq Q$. For $i \in \{0, 1\}$ let \mathcal{A}_i be a DFA for the language

$$L(p, p_i) \cap L(p_0, p_i) \cap L(p_1, p_i) \cap L(r, r_i) \cap L(r_0, r_i) \cap L(r_1, r_i).$$

Now consider the language

$$\{v_0 \# u_0 \# v_1 \# u_1 : v_i u_i \in L(\mathcal{A}_i) \text{ for } i \in \{0, 1\}, |u_0| = |u_1|, \\ u_0 \in L(q_0, p) \text{ and } u_1 \in L(q_0, r)\}$$

for which one can construct in logspace a one-counter automaton. The language above is empty if and only if \mathcal{A} has a critical tuple. \square

Lemma 7.3, Lemma 7.5 and the fact that a regular language is not streamable in space $o(n)$ in the fixed-size model if and only if it is streamable in space $O(\log n)$ in the fixed-size model implies:

Corollary 7.6. *Given a minimal DFA \mathcal{A} , one can test in nondeterministic logspace whether $L(\mathcal{A})$ is streamable in space $O(\log n)$ in the fixed-size model.*

8 Streaming algorithms for context-free languages

In this section, we show that our trichotomy result for regular languages does not carry over the context-free languages. More precisely, we show that for every $c \geq 1$ there exists a context-free language L_c such that in the

fixed-size model, L_c is streamable in space $\Theta(n^{1/c})$ but not streamable in space $o(n^{1/c})$. For this, we need the following technical result:

Theorem 8.1. *Let $t(k)$ be a monotonically increasing function and M be a linear bounded automaton which halts on input a^k after exactly $t(k)$ steps. Let $f(n)$ be the function with.⁴*

$$f(n) = \begin{cases} k & \text{if } n = (k+1)(t(k)+2) \text{ for some } k \\ 0 & \text{else} \end{cases}$$

There is a context-free language L such that $\text{space}_L^{\text{fs}}(n) \in \Theta(f(n))$.

Proof. Let Γ be the tape alphabet of M and Q the set of states of M . A configuration of M is encoded by a word from $\Gamma^*(Q \times \Gamma)\Gamma^*$ over the alphabet $\Gamma \cup (Q \times \Gamma)$. A computation of M on an input a^k ($k \geq 1$) is a sequence of configurations $c_1 \vdash_M \cdots \vdash_M c_{t(k)}$ where $|c_i| = k$ for all $1 \leq i \leq t(k)$, $c_1 = (q_0, a)a^{k-1}$ is the start configuration on input a^k , every c_{i+1} is obtained from c_i by applying a transition of M for $1 \leq i \leq t(k) - 1$, and $c_{t(k)}$ is an accepting computation. Let K be the set of all words

$$c_1 \$ \text{rev}(c_2) \$ c_3 \$ \cdots \$ c_{t(k)} \$ s \$ \text{rev}(s) \$ \text{ or} \quad (10)$$

$$c_1 \$ \text{rev}(c_2) \$ c_3 \$ \cdots \$ \text{rev}(c_{t(k)}) \$ s \$ \text{rev}(s) \$ \quad (11)$$

such that $c_1 \vdash_M \cdots \vdash_M c_{t(k)}$ is a computation on input a^k , $s \in \{0, 1\}^*$ is an arbitrary word of length k , and $t(k)$ odd (resp., even) in case (10) (resp., (11)). Notice that the word in (10) and (11) have length $(k+1)(t(k)+2)$.

For the language L from the theorem, we take the complement of K . It is not hard to see that L can be recognized by a nondeterministic pushdown automaton by guessing an error in the input word w . Possible errors are the following, where we write $w = u_1 \$ u_2 \$ \cdots u_k \$ u_{k+1}$ and the words u_1, \dots, u_k do not contain a $\$$:

- $k < 3$
- $u_{k+1} \neq \varepsilon$
- There exists $1 \leq i \leq k-1$ with $|u_i| \neq |u_{i+1}|$.
- u_1 is not the start configuration on input a^k .
- There exists $1 \leq i \leq k-2$ such that u_i is not a configuration.
- u_{k-2} is not an accepting configuration of M .
- There exists $1 \leq i \leq k-3$ odd such that $u_i \vdash_M \text{rev}(u_{i+1})$ does not hold.

⁴Since $t(k)$ is monotonically increasing, the number k in the first case is unique.

- There exists $1 \leq i \leq k - 3$ even such that $\text{rev}(u_i)u_{i+1}$ does not hold.

Since $\text{space}_K^{\text{fs}}(n) = \text{space}_L^{\text{fs}}(n)$, it remains to show that $\text{space}_K^{\text{fs}}(n) \in \Theta(f(n))$. First, we show that K is streamable in space $O(f(n))$ in the fixed-size model: If the window size n is not of the form $(k+1)(t(k)+2)$ for some k then the algorithm always rejects and no space at all is needed. Otherwise, since $t(k)$ is monotonically increasing, there is a unique number k such that $n = (k+1)(t(k)+2)$. For the window content w , the algorithm stores (i) the largest position i (encoded with $O(\log n)$ bits) such that $w[i : n]$ is a prefix of a word from K and (ii) the suffix of w of length $2(k+1) = O(k) = O(f(n))$. This allows to verify that every block has length k , that the computation is valid, and that the last block is the reverse of the second last block. Note that $t(k) \in 2^{O(k)}$ and hence $O(\log n) = O(\log(k+1) + \log(t(k)+2)) \subseteq O(k)$. Hence, the total space needed is $O(k) = O(f(n))$.

Finally, we show that $\text{space}_K^{\text{fs}}(n) \geq f(n)$ for all n . It suffices to consider a window size $n = (k+1)(t(k)+2)$ for some k . Hence, $f(n) = k$. Moreover, consider an accepting computation $c_1 \vdash_M c_2 \vdash_M \dots \vdash_M c_{t(k)}$ where $|c_1| = \dots = |c_{t(k)}| = k$. Let us assume that k is odd; the case that k is even is analogous. Now consider the 2^k many distinct words

$$w(s) := 0^{k+1}c_1 \$ \text{rev}(c_2) \$ c_3 \$ \dots \$ c_{t(k)} \$ s \$$$

for $s \in \{0,1\}^k$. The length of these words is $n = (k+1)(t(k)+2)$, which is the window size.

Consider now the space-optimal streaming algorithm \mathcal{A}_K^n , and let r be the number of states of \mathcal{A}_K^n (hence, $\text{space}_K^{\text{fs}}(n) = \lfloor \log_2 r \rfloor$). We claim that $\mathcal{A}_K^n(w(s)) \neq \mathcal{A}_K^n(w(s'))$ for all $s, s' \in \{0,1\}^k$ with $s \neq s'$. To see this, assume that $\mathcal{A}_K^n(w(s)) = \mathcal{A}_K^n(w(s'))$ for $s, s' \in \{0,1\}^k$ with $s \neq s'$. Hence, $\mathcal{A}_K^n(w(s) \text{rev}(s) \$) = \mathcal{A}_K^n(w(s') \text{rev}(s) \$)$. On the other hand, the above definition of $w(s)$ and $w(s')$ implies $w(s) \text{rev}(s) \$ \in K$ and $w(s') \text{rev}(s) \$ \notin K$, which yields a contradiction. We get $r \geq 2^k$, and thus $\text{space}_K^{\text{fs}}(n) \geq k = f(n)$. \square

Theorem 8.2. *For every $c \geq 1$ there exists a context-free language L_c such that $\text{space}_{L_c}^{\text{fs}}(n) \in O(n^{1/c}) \setminus o(n^{1/c})$.*

Proof. Consider a deterministic linear bounded automaton M that on input a^k terminates after exactly k^{c-1} steps (which is easy to construct). Thus, the mapping $f(n)$ from Theorem 8.1 satisfies

$$f((k+1)(t(k)+1)) = f((k+1)(k^{c-1}+2)) = f(k^c + k^{c-1} + 2k + 2) = k$$

and $f(n) = 0$ if n is not of the form $k^c + k^{c-1} + 2k + 2$ for some k . This implies $f(n) \in O(n^{1/c})$ and $f(n) \notin o(n^{1/c})$. Hence, by Theorem 8.1 there is a context-free language L_c with the properties stated in the theorem. \square

Let us conclude this section with two further specific examples of context-free languages.

Example 8.3. Let $L_1 = \{a^k b^k : k \geq 0\}$. This language is streamable in space $O(\log n)$ in the variable-size model. The algorithm stores (i) the current window size n and the unique numbers k, m such that $a^k b^m$ is the longest suffix of the window that belongs to $a^* b^*$. Note that $1 \leq k + m \leq n$. This information can be maintained: For a **pop**-operation, k and m are not changed unless $n = k + m$. In this case, k is decremented if $k > 0$. If $k = 0$ then m is decremented. For a **push**(b)-operation, m is incremented. Finally, for a **push**(a)-operation, the algorithm sets $k := 1$ and $m := 0$ if $m > 0$. If $m = 0$, then k is incremented.

On the other hand, we have $\text{space}_{L_1}^{\text{fs}}(2n) \geq \lfloor \log_2(n+1) \rfloor$. Similarly to Example 5.2 we can uniquely represent every number $0 \leq i \leq n$ by the bit string $\mathcal{A}_{L_1}^{2n}(a^{n+i} b^{n-i})$ of length $\text{space}_{L_1}^{\text{fs}}(2n)$, which implies $\text{space}_{L_1}^{\text{fs}}(2n) \geq \lfloor \log_2(n+1) \rfloor$. In particular, L_1 is not streamable in space $o(\log n)$ in the fixed-size model.

Example 8.4. Let L_2 be the Dyck-language over a single pair $(,)$ of brackets. We claim that $\text{space}_{L_2}^{\text{fs}}(4n) \geq n$. For this, we encode a bit string $\alpha = a_1 a_2 \cdots a_n$ ($a_i \in \{0, 1\}$) by the word $u(\alpha) = u_1 u_2 \cdots u_n$, where $u_i = ()$ if $a_i = 0$ and $u_i = (())$ if $a_i = 1$. Note that $|u(\alpha)| = 4n$ and there are 2^n strings $u(\alpha)$. Let $4n$ be the window size. We claim that $\mathcal{A}_{L_2}^{4n}(u(\alpha)) \neq \mathcal{A}_{L_2}^{4n}(u(\beta))$ for all $\alpha, \beta \in \{0, 1\}^n$ with $\alpha \neq \beta$. Otherwise, there would $\alpha, \beta \in \{0, 1\}^n$ with $\alpha \neq \beta$, say $\alpha[i] = 0$ and $\beta[i] = 1$, such that $\mathcal{A}_{L_2}^{4n}(u(\alpha)) = \mathcal{A}_{L_2}^{4n}(u(\beta))$. Let v consist of $2i - 1$ repetitions of $()$. Hence, we get $\mathcal{A}_{L_2}^{4n}(u(\alpha)v) = \mathcal{A}_{L_2}^{4n}(u(\beta)v)$. On the other hand, $u(\alpha)[2i : 4n]v \in L_2$ and $u(\beta)[2i : 4n]v \notin L_2$, which is contradiction. Hence $\mathcal{A}_{L_2}^{4n}$ has at least 2^n states, which implies $\text{space}_{L_2}^{\text{fs}}(4n) \geq n$. In particular, L_2 is not streamable in space $o(n)$ in the fixed-size model.

9 Other variable-size models

Recall that in the variable-size model the streaming algorithm processes a sequence of operations of the following types: **pop**, which removes the first letter from the current window content, and **push**(a) (for $a \in \Sigma$)⁵ which adds an a at the right end of the window. But other “variable-size models” are reasonable as well. For example a stack can be considered as the variable-size model, where symbols are popped and pushed at the right end of the current string. In general we describe a variable-size model by a subset \mathcal{M} of the following operations:

- **leftpop**: Remove the first letter.

⁵We fix a certain alphabet Σ in this section.

- **rightpop**: Remove the last letter.
- **leftpush**(a) for $a \in \Sigma$: Add the symbol a at the beginning.
- **rightpush**(a) for $a \in \Sigma$: Add the symbol a at the end.

We require that for all $a, b \in \Sigma$: **leftpush**(a) $\in \mathcal{M}$ if and only if **leftpush**(b) $\in \mathcal{M}$ and **rightpush**(a) $\in \mathcal{M}$ if and only if **rightpush**(b) $\in \mathcal{M}$. Thus, at each side of the word, either all symbols can be pushed or none. We also require that $\{\text{leftpush}(a) \mid a \in \Sigma\} \subseteq \mathcal{M}$ or $\{\text{rightpush}(a) \mid a \in \Sigma\} \subseteq \mathcal{M}$. This allows to built up words of arbitrary length. For a sequence of operations $x \in \mathcal{M}^*$ we define **string**(x) (the string computed by x) as expected, where $y \in \mathcal{M}^*$:

- **string**(ε) = ε
- **string**($y \text{ leftpush}(a)$) = $a \text{ string}(y)$
- **string**($y \text{ rightpush}(a)$) = **string**(y) a
- **string**($y \text{ leftpop}$) = **string**($y \text{ rightpop}$) = ε if **string**(y) = ε
- **string**($y \text{ leftpop}$) = w if **string**(y) = aw for $a \in \Sigma$
- **string**($y \text{ rightpop}$) = w if **string**(y) = wa for $a \in \Sigma$

A *dynamic query algorithm* for the variable-size model \mathcal{M} is then a deterministic automaton $\mathcal{A} = (S, \mathcal{S}, \Phi, s_0, F)$, where $S \subseteq \{0, 1\}^*$ is a (possibly infinite) state set, called the set of storage contents.

We say that the language $L \subseteq \Sigma^*$ is *\mathcal{M} -queryable*⁶ in space $s(n)$ if there exists a query algorithm as described above such that:

- $L(\mathcal{A}) = \{u \in \mathcal{M}^* \mid \text{string}(u) \in L\}$, and
- for all $u \in \mathcal{M}^*$ with $n = \max\{|\text{window}(v)| : v \text{ is a prefix of } u\}$ we have $|\mathcal{A}(u)| \leq f(n)$.

We can recover the variable-size streaming model considered so far as

$$\text{stream} = \{\text{leftpop}\} \cup \{\text{rightpush}(a) \mid a \in \Sigma\}.$$

Another well-known model is the stack model

$$\text{stack} = \{\text{rightpop}\} \cup \{\text{rightpush}(a) \mid a \in \Sigma\}. \quad (12)$$

The standard streaming model mentioned in the introduction corresponds to **standard** = $\{\text{rightpush}(a) \mid a \in \Sigma\}$. We remarked in the introduction that every regular language L is **standard-queryable** in constant space by

⁶We prefer the term “queryable” instead of “streamable” in this more general setting. A stack for instance is not much related to streaming.

storing only the current state of an DFA for L . If we consider now the reversed standard model $\text{standard}^{\text{rev}} = \{\text{leftpush}(a) \mid a \in \Sigma\}$, then again every regular language is $\text{standard}^{\text{rev}}$ -queryable in constant space since we can simply store and update the current state of a DFA for the reversed language L^{rev} . Moreover, for the combined model $\mathcal{M} = (\text{standard} \cup \text{standard}^{\text{rev}})$ it suffices to store the image $h(w)$ of the current string w under the syntactic morphism $h : \Sigma^* \rightarrow M(L)$. The update for a $\text{leftpush}(a)$ (resp., $\text{rightpush}(a)$) is $m \mapsto h(a)m$ (resp., $m \mapsto mh(a)$).

By the above remarks, we only have to consider variable-size models that contain **leftpop** or **rightpop** as long as we are only interested in regular languages. We call such a variable-size model *non-trivial* (it also has to fulfill the above requirements concerning push-operations). We will show that while the optimal space requirement for querying a regular language in a non-trivial variable-size model is again constant, logarithmic or linear, the corresponding partitions of the regular languages can differ from the situation for **stream**. A first observation generalizes the lower bound of $\lfloor \log_2(n+1) \rfloor$ from Theorem 4.2 to all non-trivial variable-size models (up to some additive constant). In fact, almost the same construction that we used in the proof of Theorem 4.2 covers every non-trivial model.

Theorem 9.1. *Let \mathcal{M} be a non-trivial variable-size model and $L \subseteq \Sigma^*$ a language such that $\emptyset \neq L \neq \Sigma^*$. There is a constant c such that if L is \mathcal{M} -queryable in space $f(n)$ with f monotonic, then $f(n) \geq \lfloor \log_2(n+1) \rfloor - c$ for all $n \geq 0$.*

Proof. Let us show the argument for the stack model **stack** (similar arguments work for the other cases). Assume that $\varepsilon \in L$, let $y \notin L$ be of minimal length, and let $c = |y|$ (which is a constant). Using **rightpop**-operations, we can determine n from $s(ya^n)$ (defined as in (4) with **rightpop** instead of **pop**) as in the proof of Theorem 4.2 (but we cannot compute $s(ya^n)$ from $s(a^n)$ using **rightpush**(\cdot)-operations). But still, one can encode every number $i \in \{0, 1, \dots, n\}$ by a bit string of length $f(i+c) \leq f(n+c)$, namely $s(ya^i)$. There are $2^{f(n)+c+1} - 1$ many bit strings of length at most $f(n+c)$. Hence, $2^{f(n)+c+1} - 1 \geq n+1$, which implies that $f(n) \geq \lfloor \log_2(n+1) \rfloor - c$. \square

We fix a regular language $L \subseteq \Sigma^*$ for the rest of this section. Moreover, let M be the syntactic monoid of L , $h : \Sigma^* \rightarrow M$ the syntactic homomorphism, $A = h(\Sigma)$, and $F = h(L) \subseteq M$. Recall that for every word $w \in \Sigma^*$: $w \in L$ if and only if $h(w) \in F$. Recall the definition of the colored left Cayley graph $\Gamma(M, A, F)$ from Section 2. Its vertex set is M and its edge set is $\{(x, y) \mid y = a \cdot x \text{ for some } a \in A\}$. In this section we also need a *colored right Cayley graph*. To distinguish both graphs, we denote the colored left Cayley graph $\Gamma(M, A, F)$ with $\Gamma_\ell(M, A, F)$ in the following. We define the colored right Cayley graph $\Gamma_r(M, A, F)$ as the graph with vertex set M and edge set $\{(x, y) \mid y = x \cdot a \text{ for some } a \in A\}$. The coloring is defined in the

same way as for $\Gamma_\ell(M, A, F)$: $x \in M$ is colored with 1 (resp., 0) if $x \in F$ (resp., $x \notin F$). A node $x \in M$ is reachable from $y \in M$ in $\Gamma_r(M, A, F)$ if and only if $x \leq_{\mathcal{R}} y$ in M , which is defined by

$$x \leq_{\mathcal{R}} y \iff \exists r \in M : x = y \cdot r. \quad (13)$$

Also, we have $x \equiv_{\mathcal{R}} y$ if and only if $x \leq_{\mathcal{R}} y \leq_{\mathcal{R}} x$.

We use the abbreviations $\Gamma_\ell = \Gamma_\ell(M, A, F)$ and $\Gamma_r = \Gamma_r(M, A, F)$ in the rest of Section 9.

9.1 The reversed variable-size streaming model

A first natural model is the *reversed variable-size streaming model*

$$\text{stream}^{\text{rev}} = \{\text{rightpop}\} \cup \{\text{leftpush}(a) \mid a \in \Sigma\}.$$

Based on our prior analysis, it is easy to obtain the partition of the regular languages for the reversed streaming model. If we use the same properties described in Theorem 5.4 and Theorem 5.7 for the variable-size streaming model, but on Γ_r instead of Γ_ℓ , then similar proofs cover the reversed variable-size streaming model.

Theorem 9.2. *Assume that $\emptyset \neq L \neq \Sigma^*$. Then we have:*

- *If every SCC of Γ_r is homomorphic to a directed cycle then L is $\text{stream}^{\text{rev}}$ -queryable in space $O(\log n)$ but not $\text{stream}^{\text{rev}}$ -queryable in space $o(\log n)$.*
- *If an SCC of Γ_r is not homomorphic to a directed cycle then L is not $\text{stream}^{\text{rev}}$ -queryable in space $o(n)$, but clearly $\text{stream}^{\text{rev}}$ -queryable in space $O(n)$.*

The language $a\{a, b\}^*$ from Example 5.3 is not stream -queryable in space $o(n)$ but $\text{stream}^{\text{rev}}$ -queryable in space $O(\log n)$, and vice versa for the language $\{a, b\}^*a$ (Example 5.1). Moreover, all *right open* languages, i.e., languages of the form $L\Sigma^*$ for a regular language L , are $\text{stream}^{\text{rev}}$ -queryable in space $O(\log n)$. Also note that all theorems for the fixed-size streaming model can be similarly adapted to the reversed fixed-size streaming model (which is defined in the obvious way) by using the right Cayley graph.

9.2 The stack model

In this section we consider the stack model stack defined in (12). It turns out that every regular language, which is stack -queryable in logarithmic space is also $\text{stream}^{\text{rev}}$ -queryable in logarithmic space. The other direction is not true, for example the language $(aa \mid bb)^*$ described in Example 2.1 is still $\text{stream}^{\text{rev}}$ -queryable in logarithmic space but it is not stack -queryable

in space $o(n)$. We will prove that in order to query L in logarithmic space in the stack model, all non-trivial SCCs of the right Cayley graph need to be directed cycles, and not only homomorphic to it. Recall that an SCC is trivial if it consists of a single node v and (v, v) is not an edge. We prove the upper bound for the extended stack model

$$\text{stack}_+ = \{\text{rightpop}\} \cup \{\text{rightpush}(a), \text{leftpush}(a) \mid a \in \Sigma\}.$$

Theorem 9.3. *If every non-trivial SCC of Γ_r is a directed cycle, then L is stack_+ -queryable in space $O(\log n)$.*

Proof. We store similar information as in the proof of Theorem 5.4, but for the right Cayley graph instead of the left Cayley graph. Let $w \in \Sigma^*$ be a word of length n and $\text{Pref}(w)$ be the set of prefixes of w , which includes the empty word and w itself. We define a preorder \preceq on $\text{Pref}(w)$ by $u \preceq v$ iff $h(u) \leq_{\mathcal{R}} h(v)$, where $\leq_{\mathcal{R}}$ is defined in (13). For the induced equivalence relation \equiv , we have $u \equiv v$ iff $h(u) \equiv_{\mathcal{R}} h(v)$. Let $\text{Pref}(w)/\equiv$ be the set of equivalence classes of \equiv . Note that $|\text{Pref}(w)/\equiv|$ is again bounded by a constant which only depends on the monoid M . One can identify the elements of $\text{Pref}(w)/\equiv$ with intervals on the set of positions of w , i.e., we only need to store a constant number of interval endpoints using $O(\log n)$ bits. We describe these data again by a sequence

$$p_0, m_0, p_1, m_1, p_2, \dots, m_{k-2}, p_{k-1}, m_{k-1}, p_k \quad (14)$$

such that the following holds:

- $1 \leq k \leq |M|$,
- $0 = p_0 < p_1 < \dots < p_{k-1} < p_k = n + 1$,
- $m_0, \dots, m_{k-1} \in M$ and m_0 is the unit element of M .

The equivalence classes of \equiv are the sets $C_i = \{w[1 : p] \mid p_i \leq p < p_{i+1}\}$ for $0 \leq i \leq k - 1$ (the class C_0 contains the empty prefix for $p = p_0 = 0$). The monoid element m_i is $h(w[1 : p_i])$ for $0 \leq i \leq k - 1$ (hence, $m_0 = 1$ is the unit element). Thus, $m_i = h(v)$ where v is the shortest prefix of w in its equivalence class C_i .

In order to test whether $w \in L$, one has to check whether $h(w) \in F$. We show that $h(w)$ can be computed from the monoid element $m_{k-1} = h(w[1 : p_{k-1}])$. If $p_{k-1} = p_k - 1 = n$, then $m_{k-1} = h(w)$ and we are done. Hence, assume that $p_{k-1} < p_k - 1$. Note that $m_{k-1} \equiv_{\mathcal{R}} h(w)$. Hence, the vertices $h(w)$ and $m_{k-1} = h(w[1 : p_{k-1}])$ belong to the same SCC C of Γ_r . Since $p_{k-1} < p_k - 1 = n$ we have $w \neq w[1 : p_{k-1}]$, and C must be non-trivial (it can be singleton, but it must contain an edge). By assumption, C is a directed cycle. Thus, we can compute $h(w)$ by traversing the cycle from

m_{k-1} for $n - p_{k-1}$ many steps. The color of $h(w)$ then indicates whether $w \in L$.

For a **rightpop**-operation on w and $p_{k-1} < n$, the algorithm only decrements the last position p_k since the rest of the sequence does not change. Otherwise, if $p_{k-1} = n$, then the algorithm removes p_k and the monoid element m_{k-1} from the sequence (14).

Let us now consider a **rightpush**(a)-operation on w . We already explained how to compute $h(w)$ in order to test whether $w \in L$. Now we compute $h(wa) = h(w) \cdot h(a)$. If $h(wa) \equiv_{\mathcal{R}} h(w)$, then we simply need to increment p_k . Otherwise, if we reach a new SCC in Γ_r , then we update the sequence (14) to

$$p_0, m_0, p_1, m_1, p_2, \dots, m_{k-2}, p_{k-1}, m_{k-1}, p_k, h(wa), p_k + 1.$$

Note that this operation cannot be done if the SCC is only homomorphic to a directed cycle, since we only can compute the color of $h(w)$ in this case and not the concrete element.

Finally, we consider a **leftpush**(a)-operation on w . We know that $\equiv_{\mathcal{R}}$ is a left congruence, i.e. $x \equiv_{\mathcal{R}} y$ implies $zx \equiv_{\mathcal{R}} zy$ for all $x, y, z \in M$. This means that similar to Theorem 5.4, we can obtain the interval-representation of $(\text{Pref}(wa), \preceq)$ from the interval-representation of $(\text{Pref}(w), \preceq)$ by possibly merging successive intervals. \square

Theorem 9.4. *If some non-trivial SCC of Γ_r is not a directed cycle, then L is not stack-queryable in space $o(n)$.*

Proof. Let C be a non-trivial SCC which is not a directed cycle. Then C must consist of at least two nodes (a single node with a loop is a cycle). Hence, there exists a node $c \in C$ which has successors $c_1, c_2 \in C$ with $c_1 \neq c_2$. Let $u \in \Sigma^*$ be a word such that $h(u) = c$. It follows that $c_1 = h(ua)$ and $c_2 = h(ub)$ for some $a, b \in \Sigma$ with $a \neq b$. Since c, c_1 and c_2 are in the same SCC, there are paths from c_1 and c_2 back to c . Let $u_1, u_2 \in \Sigma^*$ be the corresponding words to this paths, i.e. $h(uau_1) = h(u) = h(ubu_2)$. Choose numbers $p, q > 0$ such that $|au_1|^p = |bu_2|^q$ and let $z_0 = (au_1)^p$ and $z_1 = (bu_2)^q$. Also, since $c_1 \neq c_2$, there exist $x, y \in \Sigma^*$ such that $xuay \in L$ and $xuby \notin L$ or vice versa. Without loss of generality, assume $xuay \in L$ and $xuby \notin L$. This also means that $h(xuay) \in F$ and $h(xuby) \notin F$.

Assume now that L is stack-queryable in space $o(n)$ and consider the 2^n words $xu\{z_0, z_1\}^n$. For n large enough, there must be words $xu\alpha_1 \cdots \alpha_n \neq xu\alpha'_1 \cdots \alpha'_n$ with $\alpha_i, \alpha'_i \in \{z_0, z_1\}$ for all i such that pushing these words on the stack leads to the same storage content. Since the words are different, there is some i with $\alpha_i = z_0$ and $\alpha'_i = z_1$ or vice versa. Now we use the operation **rightpop** until the stack contains $xu\alpha_1 \cdots \alpha_{i-1}a$ and $xu\alpha'_1 \cdots \alpha'_{i-1}b$, respectively. Note that $h(u\alpha_1 \cdots \alpha_{i-1}) = h(u) = h(u\alpha'_1 \cdots \alpha'_{i-1})$. Finally, we push the word y , so the stacks contain $xu\alpha_1 \cdots \alpha_{i-1}ay$ and $xu\alpha'_1 \cdots \alpha'_{i-1}by$,

respectively. The storage contents must be still identical. On the other hand, we have $h(xu\alpha_1 \cdots \alpha_{i-1}ay) = h(xuay) \in F$ and $h(xu\alpha'_1 \cdots \alpha'_{i-1}by) = h(xuby) \notin F$, i.e., $xu\alpha_1 \cdots \alpha_{i-1}ay \in L$ and $xu\alpha'_1 \cdots \alpha'_{i-1}by \notin L$, which is a contradiction. \square

The reversed (extended) stack is covered similarly using the left Cayley graph instead of the right Cayley graph.

9.3 The full variable-size model

Finally, we consider variable-size models where it is possible to remove symbols on both sides. One of it is the *full variable-size model*

$$\text{full} = \{\text{leftpop}, \text{rightpop}\} \cup \{\text{leftpush}(a), \text{rightpush}(a) \mid a \in \Sigma\},$$

where all operations are allowed. We show that only cyclic languages are full-queryable in space $O(\log n)$. A language is cyclic, if for all $n \in \mathbb{N}$ either $\Sigma^n \subseteq L$ or $L \cap \Sigma^n = \emptyset$.

Theorem 9.5. *If L is cyclic, then L is full-queryable in space $O(\log n)$.*

Proof. It suffices to store and update the length n of the current string using $O(\log n)$ bits, because the language contains either all words of length n or none. \square

Theorem 9.6. *Assume that \mathcal{M} is a non-trivial variable-size model such that $\text{leftpop}, \text{rightpop} \in \mathcal{M}$. If L is not cyclic, then L is not \mathcal{M} -queryable in space $o(n)$.*

Proof. Assume that L is not cyclic and \mathcal{M} is a non-trivial variable-size model such that $\text{leftpop}, \text{rightpop} \in \mathcal{M}$. There are words u and v such that $u \in L$, $v \notin L$ and $k = |u| = |v|$. Assume that L is \mathcal{M} -queryable in space $o(n)$. Consider the 2^n words $\{u, v\}^n$. For n large enough, there exist words $w = w_1 \cdots w_n \neq w' = w'_1 \cdots w'_n$ with $w_i, w'_i \in \{u, v\}$ for all $1 \leq i \leq n$ such that pushing w results in the same storage content as pushing w' . Since $w \neq w'$, there exists $1 \leq i \leq n$ such that $w_i = u$ and $w'_i = v$ or vice versa. Without loss of generality, assume $w_i = u$ and $w'_i = v$. We now pop $(i-1)k$ symbols on the left and $(n-i)k$ symbols on the right. After this, the storage contents are still the same, but the current strings are w_i and w'_i , respectively. This is a contradiction, since $w_i = u \in L$ and $w'_i = v \notin L$. \square

10 Future work

It would be interesting to know the space complexity of querying regular languages in the sliding window model, when the regular language is part of the input, and, for instance, given by a deterministic finite automaton

(DFA). The syntactic monoid of $L(A)$, where A is an m -state DFA, can have size m^m [20]. This yields the space bound $O(\log(n) \cdot m^{m+1} \cdot \log(m))$ in the proof of Theorem 5.4, where n is the window size. But maybe a better algorithm exists. In particular, it would be nice to have an algorithm with a space bound of the form $O(\log(n) \cdot \text{poly}(m))$.

We have seen that the space trichotomy for regular languages does not carry over to context-free languages. But does it carry over to certain subclasses of context-free languages that properly extend the regular languages? An interesting candidate is the class of deterministic context-free languages. Note that the context-free languages constructed in Section 8 are non-deterministic. Before looking at the full class of deterministic context-free languages, one might first look at visibly pushdown languages [2] or the even smaller class of visibly one-counter languages [6, 23]. Visibly pushdown languages are defined by pushdown automata, where the current input symbol determines the executed operation on the pushdown.

Finally, one might also study weighted automata in the sliding window model. A weighted automaton computes for an input word a value from a semiring, which is the Boolean semiring for classical finite automata; see [13] for details. The goal would be to maintain the semiring value to which the sliding window content maps.

References

- [1] Charu C. Aggarwal. *Data Streams - Models and Algorithms*. Springer, 2007.
- [2] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004), Chicago, IL, USA*, pages 202–211. ACM Press, 2004.
- [3] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2004*, pages 286–296. ACM, 2004.
- [4] Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O’Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2003*, pages 234–243. ACM, 2003.
- [5] Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013.

- [6] Vince Bárány, Christof Löding, and Olivier Serre. Regularity problems for visibly pushdown languages. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science, STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006.
- [7] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Efficient summing over sliding windows. In *Proceedings of the 15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016*, volume 53 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [8] Vladimir Braverman. Sliding window algorithms. In *Encyclopedia of Algorithms*, pages 2006–2011. Springer, 2016.
- [9] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *Journal of Computer and System Sciences*, 78(1):260–272, 2012.
- [10] Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In *Proceedings of the 21st Annual European Symposium on Algorithms, ESA 2013*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013.
- [11] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.
- [12] N.G. de Bruijn. A combinatorial problem. *Nederl. Akad. Wet., Proc.*, 49:758–764, 1946.
- [13] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 2009.
- [14] Nathanaël Fijalkow. The online space complexity of probabilistic languages. In *Proceedings of the International Symposium on Logical Foundations of Computer Science, LFCS 2016*, volume 9537 of *Lecture Notes in Computer Science*, pages 106–116. Springer, 2016.
- [15] Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of the 24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [16] Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *Journal of the ACM*, 44(2):257–271, 1997.

- [17] Moses Ganardi, Danny HucKe, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [18] Paweł Gawrychowski and Artur Jez. Hyper-minimisation made efficient. In *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science 2009, MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 356–368. Springer, 2009.
- [19] Łukasz Golab and M. Tamer Özsü. Processing sliding window multi-joins in continuous queries over data streams. In *Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003*, pages 500–511. Morgan Kaufmann, 2003.
- [20] Markus Holzer and Barbara König. On deterministic finite automata and syntactic monoid size. *Theoretical Computer Science*, 327(3):319–347, 2004.
- [21] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, MA, 1979.
- [22] Richard M. Karp. Some bounds on the storage requirements of sequential machines and turing machines. *Journal of the ACM*, 14(3):478–489, 1967.
- [23] Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig. Visibly counter languages and constant depth circuits. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPIcs*, pages 594–607. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.
- [24] Michel Latteux. Languages à un compteur. *Journal of Computer and System Sciences*, 26(1):14–33, 1983.
- [25] Luc Segoufin and Victor Vianu. Validating streaming XML documents. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2002*, pages 53–64. ACM, 2002.
- [26] Jeffrey Shallit and Yuri Breitbart. Automaticity I: properties of a measure of descriptive complexity. *Journal of Computer and System Sciences*, 53(1):10–25, 1996.
- [27] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, Basel, Berlin, 1994.

- [28] Andrew Szilard, Sheng Yu, Kaizhong Zhang, and Jeffrey Shallit. Characterizing regular languages with polynomial densities. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science, MFCS 1992*, volume 629 of *Lecture Notes in Computer Science*, pages 494–503. Springer, 1992.
- [29] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science, FOCS 1977*, pages 222–227. IEEE Computer Society, 1977.